BREAKING THE TENSION: DEVELOPMENT AND INVESTIGATION OF A CENTRIFUGAL TENSIONED METASTABLE FLUID DETECTOR SYSTEM

A Thesis

by

MATTHEW ALAN SOLOM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Karen Vierow
Committee Members,	Leslie A. Braby
	James T. White
Head of Department,	Yassin Hassan

December 2012

Major Subject: Nuclear Engineering

ABSTRACT

The current knowledge of the performance characteristics of Centrifugal Tensioned Metastable Fluid Detectors is limited. While a theoretical treatment and experience with bubble chambers may be applied with some degree of success, they are no substitute for experimental and operational knowledge of real CTMFD systems. This research, as with other investigations into CTMFD systems in the past, applies theory and simulations. In addition, however, an experiment was conducted that for the first time attempts to determine the threshold energy for triggering a CTMFD system in a controlled manner.

A CTMFD system works in a manner similar to classic bubble chambers. A liquid is brought to an unstable state in which it is favorable to form a volume of vapor; using centrifugal techniques similar to those employed in a Briggs apparatus, the pressure in the sensitive region can be brought to extremely low values, placing the liquid in a tensile state. In such states, the energy necessary to cause the formation of macroscopic bubbles can be vanishingly small, depending on the degree of tension. When such bubbles form in a CTMFD, if they have a size bigger than a critical value, they will grow until a large vapor column forms in the sensitive region of the CTMFD.

The experiment developed for this research employed a carefully-controlled laser to fire pulses of known energies into the sensitive region of a CTMFD. By varying the laser power, the threshold values for the triggering energy of a CTMFD can be found.

The experiment and simulation demonstrated the ability of the facilities to test CTMFD systems and the potential to extract their operational characteristics. The experiment showed a certain viability for the technique of laser-induced cavitation in a seeded fluid, and demonstrated some of the associated limitations as well. In addition, the CFD framework developed here can be used to cross-compare experimental results with computer simulations as well as with the theoretical models developed for this research.

DEDICATION

For my grandparents.

ACKNOWLEDGEMENTS

I would like my advisor, Dr. Karen Vierow, for making this research possible. Her help, support, and above all, patience, enabled me to complete this work. I would also like to thank the other members of my committee, Dr. Leslie A. Braby and Dr. James T. White, for their help and insightful discussions on the mechanics behind the experiment.

I am also very grateful to the other members of the Laboratory for Nuclear Heat Transfer Systems, past and present, for the myriad of ways they assisted. Oren Draznin, Wes Cullum, Brad Beeny, Andrew Dercher, and Patrick McDermott were instrumental in assembling the experimental facilities. In addition, Dr. Kevin Hogan introduced me to CFD and was extremely helpful in the early stages of the CFD model development.

I would also like to thank Dr. R. Cable Kurwitz and the Nuclear Power Institute at Texas A&M University. Their support enabled me to continue after funding for this research disappeared.

NOMENCLATURE

ATMFD	Acoustic Tensioned Metastable Fluid Detector
CAD	Computer-Aided Design
CFD	Computational Fluid Mechanics
CTMFD	Centrifugal Tensioned Metastable Fluid Detector
HEU	Highly Enriched Uranium
IR	Infrared
LED	Light Emitting Diode
P	Pressure
PWM	Pulse Width Modulation
r	Radius
R	Gas Constant
SNM	Special Nuclear Material
T	Temperature
TMFD	Tensioned Metastable Fluid Detector
WIMP	Weakly Interacting Massive Particle

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

Detection of highly enriched uranium (HEU) and other special nuclear materials (SNM) can be made difficult through shielding and low intrinsic radioactivity signatures of the potential material. In addition, current detectors designed for use at domestic points of entry (for the detection of smuggled SNM) tend to be large and expensive affairs with limited usefulness. The development of an inexpensive, high-performance, and portable system would therefore greatly help the detection effort and enhance the deterrence of such activities.

A novel detector based on metastable tensioned fluids has promising characteristics that would facilitate detection of SNM in an active-interrogation environment. The detectors can be fabricated out of common, inexpensive materials, and be scaled from relatively small, portable units to large, fixed detectors by simply enlarging the tensioned fluid chamber. The threshold for detection can be adjusted by changing the degree of tension present in the fluid, allowing for the sensitivity to radiation to be easily adjusted.

There are two major classes of tensioned metastable fluid detectors currently under investigation to achieve this. One design uses a piezoelectric transducer to induce acoustic waves in a sensitive resonant region. This results in periods of high pressure at the peak of the waveform and potentially significant degrees of tension during the troughs. The detector therefore has an intrinsic duty cycle and will have a minimum

insensitive period regardless of the particle flux. Such a system is an acoustic tensioned metastable fluid detector (ATMFD).

The second design achieves a tensile state in the sensitive region through centrifugal means. A specially crafted assembly containing the fluid is rotated around its centerline at a very high rate. Due to the design of the apparatus, the result can be a significant degree of tension in the fluid at the centerline and for the region surrounding it. While the pressure in the sensitive region will vary by the distance from the centerline, it can be essentially time-independent. This eliminates the insensitive periods inherent in the ATMFD. However, this centrifugal tensioned metastable fluid detector (CTMFD) has the drawback of being essentially a single-shot device; once the tension is broken, the device needs to be stopped and restarted to re-tension the fluid. This results in dead times on the order of tens of seconds. Such a protracted period of insensitivity makes the device useless for high-flux fields within the device's sensitive range; however, it is much less important when the field is extremely weak and detection of the presence of the field takes priority over quantification of the flux.

While the conditions the fluids experience may be considered exotic, the fluids themselves need not be. The devices can be designed to use Freon, acetone, or even water. They take advantage of the tendency of liquids to adhere to the walls of their containers if the composition is compatible (adhesive forces), for individual molecules to be attracted to each other (cohesive forces), and for the liquid to remain largely incompressible. The combination of those properties allow for tensile stresses to be applied to liquids far in excess of thermodynamic stability. In fact, liquids can be

tensioned well beyond the point where the absolute pressure within them is less than zero [1]. Some liquids have remarkable limits on the amount of tension they can endure; benzene has been measured to -150 bar [2], and water has been measured to -1400 bar with a theoretical limit of -1400 to -2000 bar [3].

The ability of liquids to tolerate significant amounts of tension may seem surprising at first, but it is a known phenomenon. In fact, it is not a rare phenomenon in nature. Plants are known to take advantage of it; -15 atm can be found in the sap of redwoods, while -80 atm have been measured in creosote bushes [4].

In such tensile states, the liquids are not stable and tend to separate into liquid + vapor. However, this is limited when there are no nucleation sites available. These may be surface irregularities, suspended 'motes' in the liquid, dissolved gases, or anything else that would encourage bubble formation and growth beyond the critical size. If appropriate nucleation sites are not available, cavitation will not occur and the liquid will remain a liquid, although unstable. Increasing the degree of tension in the liquid makes it easier to destabilize and cavitate, and by increasing it far enough either the adhesive or the cohesive forces maintaining the liquid as such will be overcome and it will cavitate. Within that range, the minimum required energy for an event to result in cavitation becomes very small. Depending on the liquid, at some point the localized energy deposited by a single incident nuclear particle is enough to nucleate a bubble larger than the critical size, and it will continue to grow until the new liquid + vapor mixture reaches thermodynamic equilibrium [5]. Given enough of a tensioned liquid with even a modest

cross-section, one could build a detector based on the cavitation of the liquid. This is not unlike the classic bubble chamber, which uses superheated liquid as its working fluid.

From bubble theory, one can easily determine the energy required to grow or shrink a spherical bubble from one radius to another if the vapor and liquid pressures are known along with the surface tension of the interface. Any of the aforementioned irregularities in the liquid may provide an effective 'bubble' that can be incited to grow by local energy deposition. However, it is unclear to what extent a liquid's properties may be affected by a metastable tensile state and how influential random microscale events may be; simple application of thermodynamics and bubble theory may be insufficient for the realization of the energetics involved in a cavitation event. Experimentation is necessary to gather sufficient data on the events and to develop empirical relationships governing cavitation events.

Such experiments can be conducted by providing known energies to liquids with known tensions. This may be done by focusing a laser pulse to a small region of fairly uniform tension in a liquid and observing potential cavitation events. By adjusting the laser's properties (pulse energy, size of the region of highly focused beam energy, etc.) one can construct the relationship between energy deposition and liquid tension for laser induced cavitation.

1.1. PROBLEM STATEMENT

The broad goal of this research is the characterization of a CTMFD system through the use of:

- Theory
- An experiment utilizing laser-induced cavitation
- Computational Fluid Dynamics (CFD)

At present, there is insufficient knowledge of the behavior of fluids in real (rather than ideal) CTMFD systems to adequately characterize their responses and to optimize such systems. This research attempts to provide some of the necessary steps to address that lack of knowledge.

1.2. TECHNICAL APPROACH

The broad goals of this research can be broken into the following facets:

1. To design, construct, and operate an experimental facility for the evaluation of CTMFD systems and the collection of tensioned liquid data
2. To explore the use of laser-induced cavitation as an experimental technique
3. To investigate the energetics of a CTMFD system
4. To develop a CFD framework for modeling a CTMFD system

This research will examine tensioned liquids to increase the knowledge of metastable fluids and their behaviors in dynamic systems. An improved understanding of tensioned fluids can lead to new types of nuclear detectors based on their unique properties. Although there have been a number of experiments performed in order to determine the limiting values of tension in liquids, other properties have not been as thoroughly probed. The utility of laser-induced cavitation will also be examined as an experimental technique.

1.3. THESIS OVERVIEW

This thesis describes a multi-faceted research project, and is organized to give each facet its own section. The first section is a simple introduction. The second section provides background on TMFD systems as well as some information on the related bubble detectors. It also provides the theoretical underpinnings for CTMFD systems in discussions of tensile liquids, bubble theory, and nucleation theory.

The third section contains the largest effort in this research: the development and early operation of CTMFD experimental facilities. It is subdivided to give details on its operational theory, including discussions on Gaussian optics, RS-232 interfaces, and Pulse Width Modulation. Each component is described; many of them were designed and built specifically for this experiment. The operation of the facility is described as well, and discussion of the first set of results is included.

The fourth section describes the Computational Fluid Dynamics framework developed for this research. While the goals of this particular model are relatively

modest, it provides a foundation on which to base further research. As with the experiment, a discussion of the first set of results is included.

Any conclusions that can be drawn are discussed in the fifth section. It brings together the earlier discussions, draws conclusions, and makes recommendations for further work. Finally, the Appendices provide details that do not fit well into the main text; this includes circuit schematics, diagrams, and source code listings.

2. BACKGROUND AND THEORY

The theoretical framework for the characterization of TMFD systems has much in common with that of more traditional bubble-based detectors. They all place their working fluids in metastable states, and experience cavitation events when incident particles deposit sufficient energy in the fluid. As a result, bubble theory and nucleation theory are key to the understanding of all such systems. While these systems may all be considered superheated, they arrive at that superheat from various methods. TMFD systems achieve it by placing the liquid in a negative pressure state, so both the limits of superheat and the overall limits of tension must be considered. Methods of energy deposition are important as well, but are considered beyond the scope of work.

The key theoretical underpinnings for TMFD systems considered for this scope of work are bubble theory and nucleation theory. An understanding of tensile fluids is important as well, as is the application of such theoretical frameworks in TMFD systems and more conventional bubble detectors.

2.1. BUBBLE THEORY

For the sake of this thesis, simple bubble theory will be applied. Bubbles will be assumed to be spherical and in a quasi-static state; this allows the analysis to proceed in a simple and straightforward manner.

A spherical gas bubble at steady state in a liquid has a relationship between the inner and outer pressure that is based on the bubble radius and surface tension as given below [6].

$$P_B - P_L = \frac{2\sigma}{r} \quad (1)$$

It is clear from Eq. (1) that the internal pressure of the bubble is greater than the pressure of the bulk liquid surrounding it, and that the difference is more pronounced in small bubbles than in large ones. It also suggests that, for a given set of constant pressures, there is one specific stable bubble radius; at a smaller radius, surface tension will dominate and the bubble will begin to collapse while the opposite is true for a larger radius. It should be noted that the pressure for the bubble is the total pressure inside it; there may be several different components to the bubble's contents, and they may make a difference in the behavior of the bubble. If a component will not dissolve or diffuse away, its partial pressure will vary with bubble size and temperature, and will not be constant.

At equilibrium, the bubble will contain a partial pressure of the surrounding liquid in its vapor state as determined by the temperature of the liquid. It would also contain partial pressures of gases that are dissolved in the liquid as determined by their solubility characteristics. It may also have amounts of relatively insoluble gases contributing. Much of the distinction between the soluble and insoluble gases depends on the dynamics of the system of interest; for rapid changes in short time scales, many gases may behave as insoluble gases, but would be expected to maintain equilibrium concentrations on longer time scales or when changes are slower.

2.1.1. Limits of Quasi-Static Stability

When more information is needed, one can turn to the formulae for bubble dynamics. In general, the Rayleigh-Plesset Equation [6] is the traditional formula used for spherical bubble dynamics Eq. (2).

$$\frac{P_B - P_L}{\rho_L} = r\ddot{r} + \frac{3}{2}\dot{r}^2 + \frac{4\nu_L}{r}\dot{r} + \frac{2\sigma}{\rho_L r} \tag{2}$$

While Eq. (2) can be used for simple bubble dynamics, it can also be used to gain insight into static and quasi-static conditions. If one takes a bubble at equilibrium and perturbs it, one can determine if that equilibrium point is stable or unstable. Depending on the gases within the bubble, it may have more than one equilibrium radius. In modeling bubble dynamics, one may consider the bubble a two-component system: the vapor from the surrounding liquid, and an insoluble gas. By applying that to the Rayleigh-Plesset Equation and by using a small perturbation to shift it slightly off-equilibrium, one can determine the following:

$$P_B = P_v + P_g \tag{3}$$

$$\frac{P_v + P_g - P_L}{\rho_L} - \frac{2\sigma}{\rho_L r} = r\ddot{r} + \frac{3}{2}\dot{r}^2 + \frac{4\nu_L}{r}\dot{r} \tag{4}$$

$$P_g = \frac{nRT}{\frac{4}{3}\pi r^3} = \frac{3nRT}{4\pi r^3} \tag{5}$$

$$P_v - P_L + \frac{3nRT}{4\pi r^3} - \frac{2\sigma}{r} = \rho_L\left(r\ddot{r} + \frac{3}{2}\dot{r}^2 + \frac{4\nu_L}{r}\dot{r}\right) \tag{6}$$

$$P_v - P_L = \Delta P\,,\ \frac{3nRT}{4\pi r^3} - \frac{2\sigma}{r} = f\ ,\ \rho_L\left(r\ddot{r} + \frac{3}{2}\dot{r}^2 + \frac{4\nu_L}{r}\dot{r}\right) = g \tag{7}$$

$$\Delta P + f = g \tag{8}$$

At equilibrium, g=0. That gives a relationship between the equilibrium bubble size and the local conditions as Eqs. (9) and (10):

$$\Delta P + f = 0\,;\ P_v - P_L + \frac{3nRT}{4\pi r^3} - \frac{2\sigma}{r} = 0 \tag{9}$$

$$\left(P_v - P_L\right)r^3 - 2\sigma r^2 + \frac{3nRT}{4\pi} = 0 \tag{10}$$

Solving for the roots of Eq. (10) gives equilibrium radii. One interesting thing to note is the situation in which the liquid and vapor pressures are equal and the system is in equilibrium:

$$-2\sigma r^2 + \frac{3nRT}{4\pi} = 0 \tag{11}$$

$$r = \sqrt{\frac{3nRT}{8\sigma\pi}} \tag{12}$$

When r is greater than the value in Eq. (12), f will always be negative. Below that, f will be positive. In the limit as r gets arbitrarily large, f will approach zero. It will have a minimum in the same place as $\Delta P + f$ as given by Eqs. (13) through (16):

$$\frac{d}{dr}\left(\Delta P + f\right) = 0 = \frac{df}{dr} = \frac{d}{dr}\left(\frac{3nRT}{4\pi r^3} - \frac{2\sigma}{r}\right) \tag{13}$$

$$0 = -3\frac{3nRT}{4\pi r^4} + \frac{2\sigma}{r^2} \tag{14}$$

$$\frac{9nRT}{4\pi} = 2\sigma r^2 \tag{15}$$

$$r = \sqrt{\frac{9nRT}{8\pi\sigma}} \tag{16}$$

Equation (16) is known as the Blake critical radius [6]. On both sides, f will increase monotonically the further away from this radius a bubble is. When ΔP is greater than zero, the vapor pressure of the liquid is greater than the bulk liquid pressure, and the surface tension of the bubble is responsible for maintaining it. Plugging the Blake critical radius in to f gives a minimum value:

$$\frac{3nRT}{4\pi\left(\sqrt{\frac{9nRT}{8\pi\sigma}}\right)^3} - \frac{2\sigma}{\sqrt{\frac{9nRT}{8\pi\sigma}}} = f_{\min} = -\sqrt{\frac{128\pi\sigma^3}{81nRT}} \tag{17}$$

As a result of Eq. (17), if ΔP is greater in magnitude than $f_{\min}$ and opposite in sign, then g will always be positive and equilibrium will not be achieved. Conversely, if ΔP is less than zero, there will be one equilibrium radius. However, if ΔP is between zero and the negative of $f_{\min}$, then there will be two equilibrium radii, one on each side of the Blake critical radius.

The end result is the determination of an upper limit of stability; for equilibrium bubbles with smaller radii than the Blake critical radius, a small perturbation will result in the bubble returning to its equilibrium state. If the equilibrium radius is greater than this limit, it is unstable; a small perturbation will lead to increasing deviations from the original state. This is due to the monotonically increasing nature of f away from the Blake critical radius; if the larger bubble is perturbed, g will end up having the same sign

as the perturbation and the bubble will change size with positive feedback. On the other side of the Blake critical radius, however, g will have the opposite sign as the direction of the perturbation and it will oppose the change [6]. If ΔP is less than zero, however, the bubble will be unconditionally stable, as the sign of g will always be opposed to the direction of bubble size deviation. Of course, the preceding analysis assumed that the bubble is spherical, isothermal, and quasi-static.

From Eq. (16), it is clear that any bubble composed entirely of the vapor of the surrounding liquid will not be in a stable equilibrium state, as the critical radius of such a bubble is zero. If it has any deviation from its equilibrium size, it will either grow explosively or implode. It should also be noted that even with a nonzero quantity of insoluble gases in a bubble, the surrounding liquid can be placed under tension without affecting the stability of the bubble if it is small enough; surface tension forces can be sufficient to override a negative pressure in the surrounding liquid. Therefore, a liquid may survive being placed under tension without completely removing existing bubbles.

2.1.2. Energetics

It takes energy to grow a bubble in size. The necessary amount of work to grow a bubble quasi-statically (such that inertial and viscous effects can be ignored) can be easily calculated by integrating from r_1 to r_2, which accounts for the PV-work done at the bubble boundary. In addition, the change in energy of the bubble's contents must also be accounted for; it may not be an adiabatic process. In a growing bubble, a certain amount of the surrounding liquid will evaporate and fill the bubble to maintain a

constant partial pressure. The gases may also absorb heat from the surroundings if the process is isothermal. These relationships are expressed in Eqs. (18) through (25):

$$W_{PV} = \int_{r_1}^{r_2} \left[\left(P_L - P_v \right) + \frac{2\sigma}{r} \right] 4\pi r^2 dr \tag{18}$$

If the bubble grows from nothing to the critical radius with a constant internal vapor pressure, Eq. (18) can calculate the net energy necessary, resulting in Eq. (19) [6]:

$$W_{CR} = \frac{16\pi\sigma^3}{3\left(P_v - P_L\right)^2} \tag{19}$$

This reflection of the work may not give the entire picture; it excludes the latent heat of vaporization for the vapor and some of the work done by it. Taking more into account results in Eqs. (18) to (23). They assume that a bubble is grown from an essentially zero size sphere of initial liquid (which vaporizes) to a vapor-filled bubble at the equilibrium radius, and that the volume of liquid vaporized is negligible compared to the final bubble volume. Heat is not exchanged between the growing bubble and its surroundings; it only does PV work on them. The necessary energy to supply to that initial bit of liquid is then given by Eq. (20).

$$E_{Bubble} = W_{PV,L} + Q_{vap} \tag{20}$$

$$W_{PV,L} = \int_0^{r_c} \left(P_L + \frac{2\sigma}{r} \right) 4\pi r^2 dr = 4\pi r_c^2 \left(\frac{P_L}{3} r_c + \sigma \right) \tag{21}$$

$$Q_{vap} = n h_{vap} \tag{22}$$

$$n = \frac{4\pi P_v r_c^3}{3RT} \tag{23}$$

In Eq. (22), the latent heat of vaporization should be expressed in a per mole basis, allowing a simple application of the Ideal Gas Law in the calculation.

$$E_{bubble} = 4\pi r_c^2 \left(\frac{P_L}{3} r_c + \sigma \right) + \frac{4\pi P_v r_c^3}{3RT} h_{vap} \tag{24}$$

$$E_{bubble} = \frac{16\pi\sigma^3}{3(P_v - P_L)^3} \left[P_v \left(3 + \frac{h_{vap}}{RT} \right) - P_L \right] \tag{25}$$

Substitution of the critical equilibrium radius into Eq. (24) results in the value given in Eq. (25). The values expressed in Eqs. (18) through (25) reflect reversible processes; if there is an irreversible component or energy loss, it must also be taken into account. In addition, they say nothing about the source or availability of the energy. For this, some discussion of bubble nucleation is necessary.

2.1.3. Acoustic Dynamics

Although the primary focus of this research involves static methods (constant rotation rates and pressure fields, etc.), some discussion of more dynamic behavior is in order. As a common experimental technique for the investigation of cavitation and tensile liquids involves acoustic waves, some consideration of their effects on bubble stability is worth mentioning.

One important aspect of bubbles exposed to acoustic waves is their behavior approximating a damped oscillator if the pressure wave's amplitude is low. This gives rise to a resonance frequency for the bubble, which is approximated in Eq. (26) [7]. In it, η is the shear viscosity and κ is the polytropic index, while r_0 is the equilibrium bubble radius and P_0 is the constant term in the external pressure oscillation.

$$\omega_0 = \frac{1}{r_0\sqrt{\rho}}\sqrt{3\kappa\left(P_0 + \frac{2\sigma}{r_0} - P_v\right) - \frac{2\sigma}{r_0} + P_v - \frac{4\eta^2}{\rho r_0^2}} \tag{26}$$

In the undamped case, expressions for the resonance frequency and bubble radius can be seen in Eqs. (27) and (28) [8].

$$\rho\omega_r^2 r_0^2 = 3\gamma\left(P_0 + \frac{2\sigma}{r_0}\right) - \frac{2\sigma}{r_0} \tag{27}$$

$$\Delta r = \begin{cases} \dfrac{P_A}{\rho r_0\left(\omega_r^2 - \omega^2\right)}\left[\sin(\omega t) - \dfrac{\omega}{\omega_r}\sin(\omega_r t)\right] & \omega \neq \omega_r \\ \dfrac{P_A}{2\rho r_0 \omega^2}\left[\sin(\omega t) - \omega t\cos(\omega t)\right] & \omega = \omega_r \end{cases} \tag{28}$$

Another important relationship for bubbles exposed to pressure oscillations is the limit of stability for such systems. In cases where gas diffusion can play a role, an expression for the amplitude limit can be seen as Eq. (29) [8], where δ is a damping factor, the gas concentrations are C_0 at saturation and C_∞ in the bulk liquid, and β is the fraction of the resonance frequency.

$$\left(\frac{\Delta P}{P_0}\right)^2 = \frac{3}{2}\left(\frac{1 + \dfrac{2\sigma}{r_0 P_0} - \dfrac{C_\infty}{C_0}}{1 + \dfrac{2\sigma}{r_0 P_0}}\right)\left[\left(1 - \beta^2\right)^2 + \beta^2\delta^2\right] \tag{29}$$

$$\delta = \frac{P_0}{\rho\omega^2 r_0^2} \tag{30}$$

$$\beta = \frac{\omega}{\omega_r} = \sqrt{\frac{\rho\omega^2 r^2}{3\gamma P_0}} \tag{31}$$

A simpler, alternate expression for the value in Eq. (29) can be seen when the concentration of gas dissolved in the liquid (C_∞) is equal to its saturation value (C_0), and the damping factor is zero. It then can be approximated as Eq. (32) [8].

$$\left(\frac{\Delta P}{P_0}\right)^2 = \frac{3\gamma\sigma}{r_0 P_0}\frac{\left(1-\beta^2\right)^2}{\left(1-\frac{\beta^2}{8}\right)} \tag{32}$$

2.1.4. Cavitation Damage

Bubble growth may be more relevant to this research, but the collapse of bubbles bears mentioning as well. Whether the source of nucleation is heterogeneous or homogenous, the appearance of bubbles in liquid may pose challenges to engineers. In transient conditions, the bubbles may only exist for moments and can collapse quickly and violently. Such collapsing bubbles can have powerful effects; they may produce local high-amplitude shock waves and microjets [6]. Heating in the collapsing bubble may result in sonoluminescence. Incandescence and thermochemical reactions have been suggested as mechanisms for light emission from collapsing bubbles, but the phenomenon is not fully understood [7]. In fact, it has been suggested that collapsing bubbles can get hot enough to allow for thermonuclear reactions [5], although that position is not widely accepted.

In any case, if any objects are near violently collapsing bubbles, they may experience cavitation damage. Such damage has been observed on boat propellers, and all sorts of hydraulic equipment are susceptible. It is a limiting condition for pump operation, and is the cause of the Tarbela tunnel collapse in Pakistan in 1974 [1].

Therefore, it is a concern to those working on equipment that can be put into such an environment.

2.2. NUCLEATION THEORY

Bubble nucleation will occur preferentially where discontinuities exist in the liquid that provide weak spots [9]. This can be observed in the simple case of boiling a pot of water on a stove; the bubbles will largely form where the water meets the heated part of the pot. If that surface is scuffed or has other imperfections, those sites themselves can be observed to be active sources of steam bubbles [9]. This is exacerbated by the local thermal conditions; the water in the immediate vicinity of the heated part of the pot will tend to be hotter than the bulk liquid, and may be fairly superheated. That encourages additional nucleation in those regions.

If such nucleation sites are not available, a liquid will be able to survive a significant amount of superheat without boiling. This can be seen by placing a smooth mug of relatively pure water in a microwave oven [10]. If the conditions are right, the water may be heated well above its boiling point without actually boiling. If it is then removed from the microwave oven, it can be hazardous; the introduction of nucleation sites may lead to sudden vigorous boiling that can cause injury [10].

2.2.1. Heterogeneous Nucleation

The discontinuities necessary for heterogeneous bubble nucleation in a liquid can be provided by the liquid's container. The container wall itself not only provides a discontinuity, but may also provide locations where gases can become trapped. If there are crevices in the wall, gas can easily be retained in them when the container is filled

with liquid. Under the right conditions, a stable equilibrium can be achieved. If the pressure is reduced, the trapped gas can expand and may release bubbles into the bulk liquid. Such trapped gas can be eliminated by pressurizing the liquid to pressures on the order of 1000 bar and holding there for half an hour or more; this forces the gas to dissolve into the liquid [1].

Rough container walls are not the only locations that one may find gas-storing crevices. Motes (particles, often specks of dirt or dust) and contaminants suspended in the liquid are also prime locations for gas-trapping crevices. In addition, if the suspended particles encounter each other, bubbles can form through tribonucleation [1].

It is possible for gas trapped in such crevices to remain in the crevice instead of dissolving into the liquid. If the wall does not get perfectly wetted and the contact angle is correct, it can reduce the gas pressure in the crevice to that of the equilibrium pressure from the gas already dissolved in the liquid. Changes in the solubility of additional gas or in the liquid pressure can upset this equilibrium.

Surface tension and geometry are not the only ways to keep gas bubbles from dissolving out into the liquid. In fact, small bubbles of gas may stay dispersed in the liquid without vanishing. Such bubbles are an additional source of weaknesses in the liquid, and their persistence was unexpected. It is now thought that even trace amounts of an organic contaminant in the liquid is sufficient to form a sort of skin around the edges of microbubbles, limiting the diffusion of gas into the liquid [6].

2.2.2. Homogeneous Nucleation

When heterogeneous nucleation is limited, homogenous nucleation may dominate the formation of vapor bubbles. Given a critical radius, there is a finite probability that a bubble will randomly form within a set time period. Since the rate of bubble formation is generally expressed as being proportional to Eq. (34) [6], the rate of formation of bubbles in a mole of tensile liquid can be given as Eq. (36) [11]. This can be adjusted to give the limit of tension when a bubble forms in a mole of tensile liquid as a function of time as in Eq. (37); in practice, this relationship can be simplified to Eq. (38) [11]. Other analyses use the critical temperature in Eq. (33), and it is generally accepted that when $Gb < 11.5$, cavitation is assured [6].

$$Gb = \frac{W_{cr}}{kT} \tag{33}$$

$$J = J_0 e^{-Gb} \tag{34}$$

$$J_0 = N\sqrt{\frac{2\sigma}{\pi m}} \tag{35}$$

$$\frac{dn}{dt} = \frac{NkT}{h}\exp\left(-\frac{\Delta f_0^* + W_{\max}}{kT}\right) = \frac{NkT}{h}\exp\left(-\frac{\Delta f_0^* + \frac{16\pi\sigma^3}{3P^2}}{kT}\right) \tag{36}$$

$$P_t = -\sqrt{\frac{16\pi}{3}\frac{\sigma^3}{kT\ln\left(\frac{NkTt}{h}\right) - \Delta f_0^*}} \tag{37}$$

$$P_t = -\sqrt{\frac{16\pi}{3kT}\frac{\sigma^3}{\ln\left(\frac{NkT}{h}\right)}} \tag{38}$$

Such nucleation has the practical effect of limiting the tensile strengths of liquids. The tensile strength of a liquid can be estimated by envisioning the intermolecular spaces as existing vapor bubbles and determining the tension in the liquid that would cause those bubbles to grow. Unfortunately, such an estimate would be incorrect, frequently missing the mark by several orders of magnitude. It would, however, give results comparable to those obtained by using compressibility moduli [6]. A better, although still inaccurate, estimate can be made by application of the van der Waals Equation, shown as Eq. (39) [1].

$$\left(P + \frac{a}{V^2}\right)(V - b) = RT \tag{39}$$

Tracing an isotherm in the van der Waals Equation reveals a minimum pressure in the curve, which is the limiting low pressure at that temperature [1]. Example isotherms are shown in Figure 1. Additionally, there is a local maxima in the curve which represents the greatest pressure a supercooled vapor can survive without condensing. By following the minimum pressure at differing temperatures, one can generate the liquid spinodal curve. Likewise, tracing the location of the local maximum, one can arrive at the vapor spinodal curve. These curves meet at the critical point, and only exist below it [6]. When the saturation curve is overlaid on the spinodal curves, one can see the ranges of the supercooled vapor and superheated liquid. The area between the saturation curve and the spinodal curves on a PV-diagram are the metastable

states for the supercooled vapor and superheated liquid. Since they are not stable, disturbances in those states can cause a transition from a single-phase system (liquid or vapor) to a two-phase system (liquid + vapor).

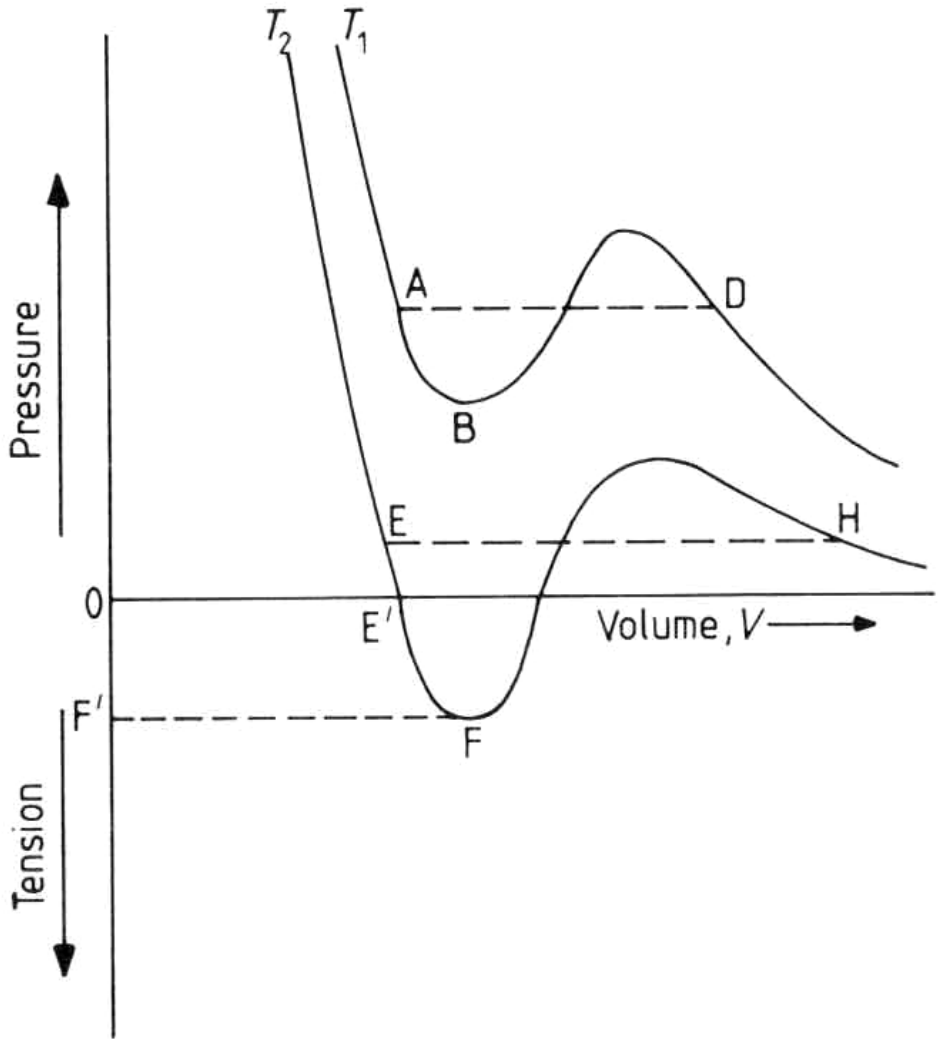

Figure 1: van der Waals Isotherms [1]

Formally, a thermodynamic spinode is defined in Eq. (40) [12]. The spinode exists where, in an isotherm, the partial derivative of the pressure with respect to the volume of the material is zero.

$$\left(\frac{\partial P}{\partial V}\right)_T = 0 \tag{40}$$

In reality, there are deviations from the forms of the spinodal lines given by the van der Waals Equation; an example curve is given in Figure 2. On a P-T plot, as the temperature decreases away from the critical point, the liquid spinodal line is expected to decrease monotonically, but this may not be the case for water. Water, a material known for behaving in non-ideal manners, is thought to be especially deviant along its liquid spinodal line, and may not exhibit monotonic behavior [13]. Furthermore, it is not achievable in practice to bring a metastable fluid all the way to its spinodal limit. The closer to the limit one approaches, the more likely it is for a random disturbance to cause nucleation and form a two-phase system.

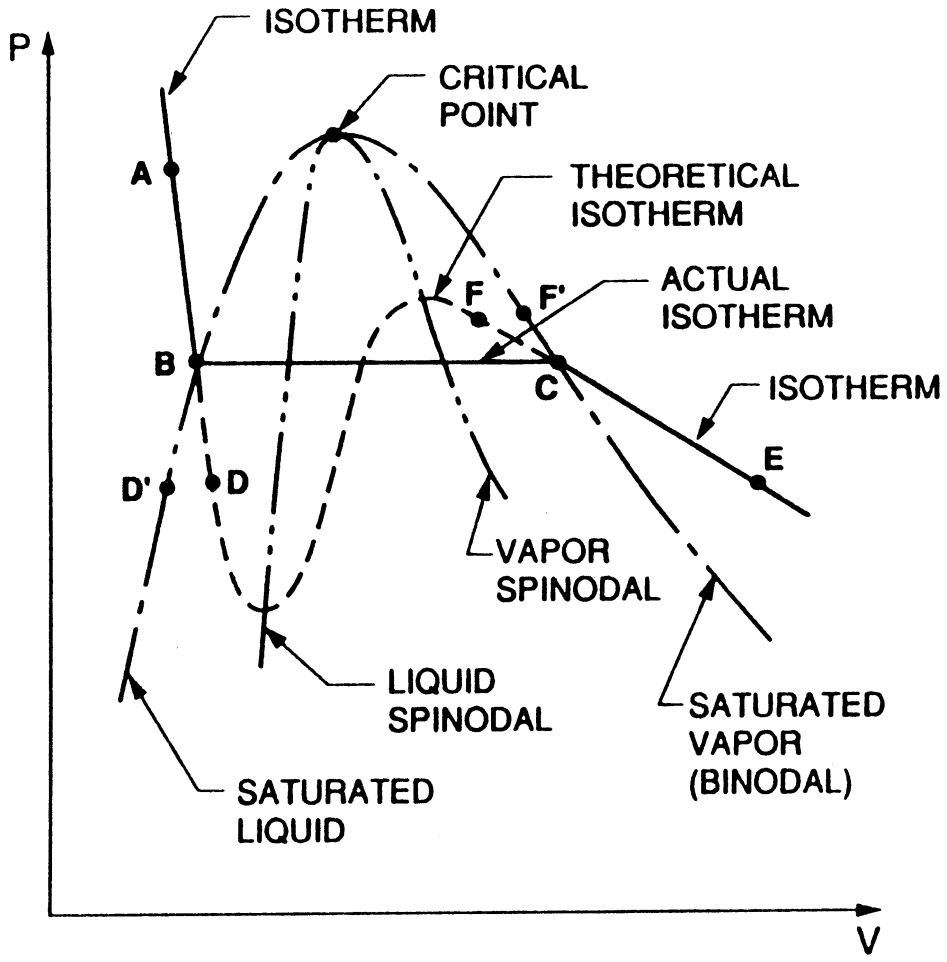

Figure 2: PV Diagram [6]

Homogenous nucleation theory can give an insight into practical limits to metastable states. Given a critical radius for a vapor bubble, there is a non-zero probability that a vapor bubble at least that large will spontaneously coalesce in a given time for a volume of metastable liquid. The closer to the spinodal limits one approaches, the shorter the waiting time becomes for the random nucleation of a critical-sized

bubble. Ultimately, a metastable liquid cannot exist as such beyond the spinodal limits, and nucleation is assured.

2.3. TENSILE FLUIDS

Much like solids, liquids are able to withstand being "stretched," or placed under a degree of tension. At first glance, this may seem a bit strange; when the pressure above a liquid drops below its vapor pressure, it may begin to boil. Indeed, this can be observed by heating a pot of water on a stove; once the water's temperature is high enough, its vapor pressure is equal to the pressure of the surrounding air and the pot will begin to boil. This is a well-known phenomenon; the boiling point of water is common knowledge, and boiling water is a part of everyday life.

However, it is not always the case that hot water at or above its boiling point will end up boiling. A mug of water heated in a microwave oven may remain as a liquid well above its boiling point; this is known as a superheated liquid, and can give an unlucky observer a close and personal demonstration of superheated water flashing to steam [10]. Such a situation demonstrates two things: liquids can exist as liquids above their boiling points, and such superheated conditions are not necessarily stable. A superheated liquid, therefore, is metastable; it can exist in such a state, but if it experiences a sufficient disturbance, it will move toward its stable equilibrium state.

The reason for this is simple: each molecule of the liquid has some attractive and repulsive forces between it and its neighbors, and the attractive forces can be quite large; these forces are large enough that liquids can be considered to be incompressible. The attractive forces between molecules in the liquid are called cohesive forces, and the

attractive forces between the liquid and its container are called adhesive forces. At fluid interfaces, these forces are agglomerated and called the surface tension.

At the boiling point, the cohesive forces in a liquid can be great enough to prevent bulk boiling, and it readily occurs only at weak points within the fluid. These weak points can be provided by bubbles, suspended particles, contaminants or other such discontinuities.

One result of this is the ability to maintain liquids in superheated or other metastable states. In fact, it can be such a powerful effect that liquids can be maintained in metastable states where their vapor pressures are much greater than the total pressure in the liquid; the total pressure in the liquid can even drop below zero, putting the liquid in a tensioned rather than compressed state.

The degree of tension that a metastable tensioned liquid can survive can be surprisingly high. For example, by 1951 benzene had been brought down to -150 bar, aniline had been brought down to -300 bar, and chloroform had been brought down to -317 bar [2]. Water in particular can survive a remarkable degree of tension; it has been tested to be metastable at pressures less than -1400 bar, with a theoretical limit between -1400 bar and -2000 bar [3]. Limits on tension can be approximated based on overcoming the attractive forces between individual molecules [6]; however, this method tends to overestimate the magnitude of the tensile limits by several orders. Better estimates can be determined by application of nucleation theory [6].

Modern studies of tension in liquids began in the 19th Century; Euler predicted the presence of tensile stresses in moving liquids in 1754 [1]. François Donny in the

1840s experimented with vacuum pumps and sulfuric acid in U-tubes, producing mild tensions in the liquid. Soon afterward, Berthelot in the 1850s was able to induce tension at 50 atm in water using an instrument that now bears his name: the Berthelot tube [1]. A Berthelot tube is a sealed cylinder filled with liquid and a small portion of air. The tube is heated, and the liquid expands until there is no room left for the air, which then dissolves in the liquid. The tube is then cooled, and tension builds up in the liquid until the tension breaks, and some of the air comes out of solution as bubbles. There is also a slight increase in the volume of the tube from just before the breaking of the tension to afterwards; this is used to help determine the degree of tension at that point [1]. Then, in the late 19th Century, Reynolds experimented with centrifugal means for tensioning liquids [1].

While tensile liquids at first glance seem to be a laboratory curiosity or limited to transient situations due to their metastability, they do appear in nature. In the 1890s, it was proposed that tension in the sap of tall trees enabled it to be lifted to the top; a vacuum pump cannot lift water anywhere near 100-m above ground, which is a height achieved by redwoods [4]. Using a balancing pressure technique, the pressure in the xylem at the top of an 82-m redwood was measured to be less than -15 atm. Haplophytes ranged from -35 to -60 atm, and a creosote bush was found with a pressure lower and -80 atm in the xylem. Mistletoe was found to have significantly lower pressures than its host, often by 10 to 20 atm [4]. The balancing pressure technique has been compared to other methods with mixed results. Experiments using modified pressure probes frequently do not agree with the high degree of tension found by the

balancing pressure method. However, centrifugal methods have been found to agree with the balancing pressure method [14].

2.4. BUBBLE CHAMBERS

The idea that nuclear particles can nucleate bubbles, and that such a process can be used as a detector, is not new. Glaser experimented with superheated diethyl ether in the early 1950s, and found that it would erupt in boiling much quicker when in the presence of ionizing radiation than in the absence of radiation [15]. From those early experiments, the early bubble chambers were developed and their theories of operation determined. Fluids such as liquid hydrogen and liquid propane were used with high fractions of superheat; a liquid hydrogen bubble chamber operating at 27 K has a superheat of nearly 7 K (with a boiling point of 20.3 K at atmospheric pressure, this is more than 1/3 greater than the boiling point), and can have a bubble nucleated by a local deposition of 4.08 eV [16].

Those systems would be operated on a cycle; the detector would be decompressed to a very superheated state in which the liquid would be very sensitive to deposited energy. Incident particles would leave tracks made of bubbles, which would be photographed. Then a fast compression would bring the liquid back to a stable state in which it would not be sensitive. Then the cycle repeats, with a decompression down to very superheated states [17]. Such devices had drawbacks resulting from the severe amounts of superheat applied to the working liquid, and the sensitive period could only last a few moments [17]. They eventually disappeared from common use, but were not completely forgotten. The COUPP (Chicagoland Observatory for Underground Particle

Physics) experiments have shown a renewed interest in similar devices for the detection of Weakly Interacting Massive Particles (WIMPs). The superheat is more moderate, and the fluid is different (CF_3I). The COUPP experiments have had a degree of success taking the sensitive period for the system from moments to indefinitely long periods [17].

Other modern takes on superheated liquids include emulsion-based devices. Small droplets of superheated liquid are suspended in an inert matrix; the two fluids are immiscible and cannot mix. Deposition of sufficient energy can result in the formation of a vapor bubble within a suspended droplet; the entire droplet may completely flash to vapor. Such devices have been used as neutron detectors, but are also sensitive to heavy charged particles [18].

While traditional bubble-based detectors involve superheat at positive pressures, speculation on devices very much like modern acoustic TMFD systems goes back at least to 1965. Bertolotti, Sette, and Wanderlingh considered the use of ultrasonic pressure waves in resonant chambers to grow microbubbles left behind by incident nuclear particles. If microbubbles were to dissipate slowly and accumulate as the result of energy deposition from incident radiation, induced ultrasonic cavitation might be able to reveal a spectrum of energy deposition long after the passage of the depositing particles [19]. While the descriptions they gave are too limited to draw conclusions about their work, the system described appears to have a remarkable amount in common with more modern ATMFD designs.

2.5. TENSIONED METASTABLE FLUID DETECTOR SYSTEMS

A novel take on the classic bubble chamber for detecting nuclear particles is the Tensioned Metastable Fluid Detector (TMFD). In such systems, a liquid is placed under tension to create a sensitive zone. The idea is that nuclear particles incident in the sensitive zone may deposit enough energy in the liquid to nucleate a bubble larger than the local critical size, at which point the bubble will continue to grow until some external limit is reached. Depending on the fluid properties and the degree of tension, the necessary deposited energy from such a particle can be quite small. The fluid need not be exotic; selection of the fluid may be a part of the design optimization process. This depends on the type, energy, and flux of particles to detect, as well as outside factors such as cost and availability.

There are two main methods for achieving the necessary tensile state in the liquid, and these methods define the class of TMFD systems. Both methods are deceptively simple. One method uses transducers to induce acoustic waves in the detection fluid; such systems are Acoustic TMFD (ATMFD) systems. The second method rotates a special assembly rapidly around an axis so that centrifugal effects create a tensile region in the vicinity of the axis of rotation; this method is used in Centrifugal TMFD (CTMFD) systems.

Both ATMFD and CTMFD systems can be made of commodity hardware and have no requirement for exotic or expensive materials; even prototype systems can be relatively inexpensive to design and construct. They can also be scaled from very small

systems to much larger ones. This may be simpler for an ATMFD than for a CTMFD, as the CTMFD must rotate the entire detection assembly.

In an ATMFD, a resonant chamber is filled with a chosen detection fluid. Piezoelectric transducers are used to induce acoustic waves in the fluid with a large amplitude. At the peaks of the acoustic waves, the pressure can be very high in the surrounding fluid; it is unlikely to be sensitive to incoming nuclear particles in this part of the cycle. The pressure will drop with the wave, and near the troughs the pressure in the surrounding liquid can be very low; given a great enough amplitude, the troughs can place the local liquid in a tensile state. At that time, the surrounding liquid can be sensitive to incoming particles. During this period, if incident particles deposit enough energy, cavitation events can occur and the resulting bubbles can quickly grow. However, when the pressure rises, the bubble growth will be arrested. The period of high pressure can cause the bubbles to violently collapse; the resulting click and pops may be audible, and cavitation luminescence may be observed. If the ATMFD is instrumented, the time and location of collapsing bubbles can be determined to help characterize the incoming flux. While the sensitivity of the detector is time and position-dependent, it can handle numerous simultaneous events and can operate continuously.

A CTMFD is similar to devices used by Briggs experimenting with tensile liquids [1] [20]. The heart of a CTMFD is the specially-shaped detection assembly. It has a bulb at the bottom along the axis of rotation; this region contains the tensioned liquid. From the top of the bulb, two tubes extend up and out radially, then bend back toward each other, meeting again along the axis of rotation some distance above the

bulb. The tubes trace a roughly diamond-shaped area above the bulb [5]. In operation, the bulb and tubes will be filled with liquid to a point above the 'elbows' in the tubes where they bend back toward each other above the bulb, but not all the way to where they meet. This is important; when the assembly is rotated, the tension at the centerline depends on how far out from the axis of rotation the liquid is filled to.

When the detection assembly in a CTMFD system is rotated rapidly about its axis, the pressure in the centerline depends on three primary things: the density of the liquid, the rotation rate, and the radial distance to the centerline from the top of the liquid fill. The pressure at the air-liquid interface (the assembly is open at the top) provides a necessary reference. Away from the centerline, the pressure in the liquid depends on the radial position. An illustration of the CTMFD concept is given in Figure 3.

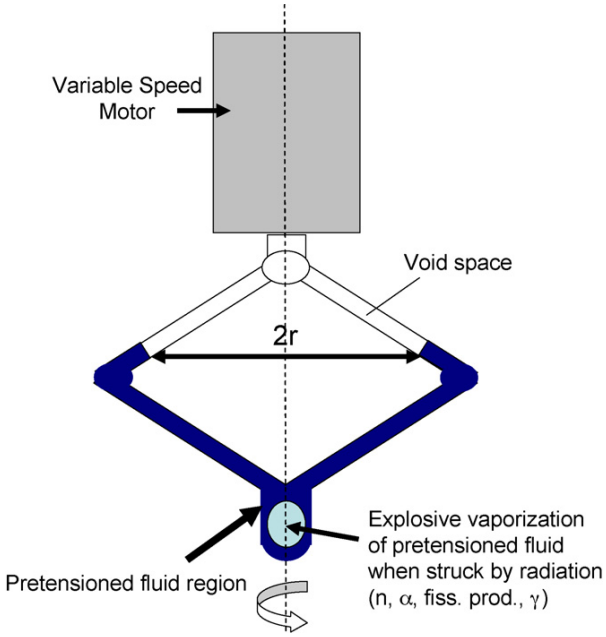

Figure 3: CTMFD Concept [5]

During operation in a CTMFD system, the assembly is typically rotated at a constant rate. As a result, the pressure field in the liquid is essentially time-independent. It is also very predictable. However, cavitation events are not detected in the same manner as in an ATMFD, and this provides a limit on the performance of a CTMFD system. When a bubble that is larger than the critical size is formed, it quickly grows and merges with any other nearby bubbles. Because there is no pulse of high pressure, the bubble's growth will not be arrested by it and it will not collapse. Instead, it moves to the centerline and grows into a vapor column in the bulb; the growth is finally stopped by the increase in pressure brought by the displaced liquid pushed up into the tubes above the bulb. An illustration of this is shown in Figure 4. Here, a "triggered" CTMFD is one that has, in operation, experienced a critical cavitation event and as a result has a stable vapor column formed in the bulb.

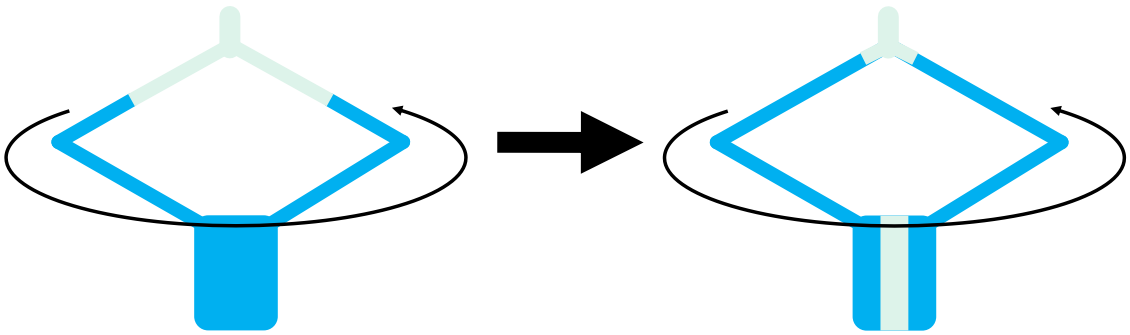

Figure 4: Triggered CTMFD

The formation of a vapor column will break the tension in the liquid; the vapor column can shrink and grow to balance the pressures. Therefore, a CTMFD is essentially a single-shot detector. Once the vapor column forms, it does not disappear. Instead, the rotation must be stopped. At that point, the vapor column will shrink to a much smaller bubble or disappear entirely. Whatever bubble remains must be allowed to migrate up through the tubes and out to the interface between the liquid and air in the upper part of the assembly. Otherwise, when the assembly is returned to speed, the bubble will regrow into the vapor column. The stop-restart sequence can take tens of seconds, so a CTMFD is not suitable for high detection-rate environments.

One of the more interesting features of a TMFD is the potential ability to reveal the directionality of incoming particles. If an incoming particle does not lose too much of its energy in an encounter in the fluid, does not deflect far from its original course, and has sufficient distance between induced nucleation events, the formation of bubbles from the high-energy depositions from the particle can provide a track for the particle. With high capture speeds and sufficient resolution, such tracks can be recorded and be used to determine the direction from which the particles were coming.

The idea behind a CTMFD is displayed in Figure 3. The main driver behind the pressure gradient is the differing fill levels in the upper and lower arms during rotation. Since the gas above the liquid is generally going to be much less dense, the pressure developed by it is going to be significantly less in the upper arms than that developed by the liquid in the lower arms. So long as the liquid adheres to the walls of the container and does not develop bubbles, the liquid will remain trapped in the lower arms and bulb.

Assuming incompressibility and neglecting gravity, the pressure drop from one radial position to another in the liquid is easily calculated as is done in Eq. (41). If it is assumed that at the liquid-gas interface the pressure P_{amb} is equal to the outside pressure, then one can approximate the centerline pressure in the bulb from the liquid's density, rotation rate, fill level, and ambient pressure as in Eq. (42) [5].

$$\Delta P = -\int_{r_1}^{r_2} 4\rho\pi^2 f^2 r dr \tag{41}$$

$$-P_{centerline} \approx 2\rho\pi^2 f^2 r_{amb}^2 - P_{amb} \tag{42}$$

While both physical demonstrations and computer simulations of CTMFD systems have been performed [5], there is a current lack of hard experimental data in such systems. In particular, the threshold deposition energies for triggering a CTMFD have not been experimentally verified. Theory and computer simulations are wonderful tools, but they cannot completely replace experimental testing and verification.

3. EXPERIMENTAL WORK

An experimental CTMFD system was designed and constructed with the goal of examining the energetics of a CTMFD system. In order to do this, it takes a CTMFD system and attempts to induce cavitation by depositing energy in the working fluid with laser pulses. The laser pulses are at known intensities and durations, and focused near the centerline in the CTMFD. The laser power starts of low, and then grows sequentially in order to find the minimum threshold power necessary to induce cavitation and the formation of a vapor column in the CTMFD.

The goals of this experiment were to examine the threshold energy for the formation of a vapor column in a seeded CTMFD with pressures in the -1 atm to -5 atm range. The liquid was acetone, and the seeding material was printer toner. A side-goal of this is to demonstrate the ability to successfully assemble a CTMFD system using standard, inexpensive parts and components. While most of the electrical systems were not off-the-shelf pieces of equipment, they use standard and relatively inexpensive components (transistors, diodes, op-amps, etc.) that a serious electronics hobbyist may be familiar with. Some circuits were made of components found entirely in local RadioShack retailers. However, there were some exceptions. The CTMFD test section used glassware that was custom-built by an experienced glassblower, and is not something that can be readily found or assembled from commercial products. In addition, the laser system's cost dwarfed that of the other pieces of equipment.

The design and construction of the experimental facilities involved a major effort. While the particular goals of this experiment are somewhat limited, the now-existing facilities provide a framework for further tests without the need to develop completely new facilities; they can largely be used as-is.

3.1. EXPERIMENTAL BACKGROUND THEORY

The idea behind the experiment is simple: operate a CTMFD in a known configuration, and focus pulses of known laser energy into a spot in the immediate vicinity of the centerline of the CTMFD, and see whether or not it forms a vapor column. If one did not form, the laser pulse energy would be increased until a threshold energy was found that did result in such a formation. With the optical system in place, the energy concentration at the focus point could be estimated, and with properties such as the Beer-Lambert coefficients, the total energy absorbed at that point could also be estimated. Using a calculation based off of the rotation rate, fill level, and density of the liquid under examination, a relationship could be drawn between the operating centerline pressure and the minimum deposition energy needed for the formation of the vapor column.

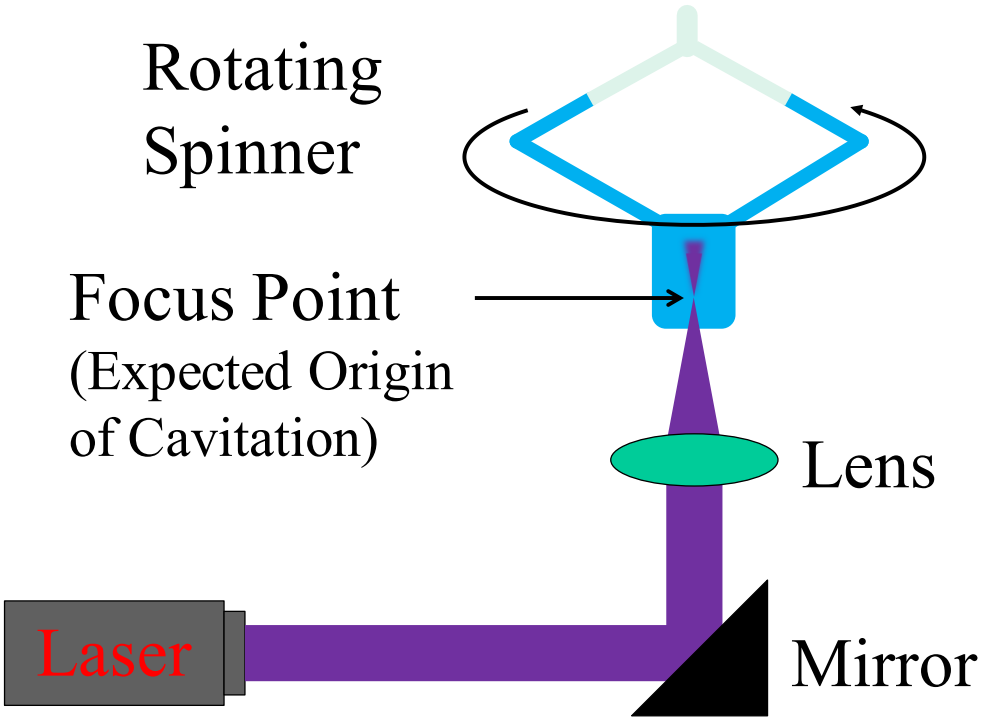

Figure 5: Experimental Concept

The size of the laser spot would not be varied; only the pulse power and duration would be. In a ramp test, the duration would be preset, and the pulse power would be the only thing varied. Since the laser spot was expected to be much larger than the critical radius of a vapor bubble in the tensile liquid, much of the deposited energy would be wasted; only a fraction would be used to nucleate the critical bubble.

The minimum reversible energy for the formation of a bubble of with a critical radius is given from Bubble Theory as Eq. (19); Eq. (25) may also be applied. An irreversible term can be added as well to describe off-equilibrium energy losses, but that is not done here. The entirety of this energy must be deposited in the immediate vicinity of the growing bubble. If this area is limited to a sphere the size of the condensed liquid

that would be the vapor fill of a critical bubble, the energy arrives in the form of a photon flux, and a known fraction of the flux incident on the circle defined by the projection of the liquid sphere into two dimensions, then, using the time duration of the flux, an estimate of the laser energy can be made by following Eqs. (43) through (54).

$$A_{dep} = \pi r_{condensed}^2 \tag{43}$$

$$r_{condensed} = \sqrt[3]{\frac{3}{4\pi} V_{condensed}} \tag{44}$$

$$V_{condensed} = \frac{m_L}{\rho_L} = \frac{nM_L}{\rho_L} = \frac{M_L}{\rho_L}\frac{P_v V_{crit}}{RT} \tag{45}$$

$$V_{crit} = \frac{4}{3}\pi r_c^3 = \frac{4\pi}{3}\left(\frac{2\sigma}{P_v - P_L}\right)^3 \tag{46}$$

$$V_{condensed} = \frac{M_L}{\rho_L}\frac{P_v}{RT}\frac{4\pi}{3}\left(\frac{2\sigma}{P_v - P_L}\right)^3 \tag{47}$$

$$r_{condensed} = \sqrt[3]{\frac{M_L P_v}{\rho_L RT}}\left(\frac{2\sigma}{P_v - P_L}\right) \tag{48}$$

$$A_{dep} = \pi\left(\frac{M_L P_v}{\rho_L RT}\right)^{2/3}\left(\frac{2\sigma}{P_v - P_L}\right)^2 \tag{49}$$

$$\Phi = \frac{Energy}{Area} \tag{50}$$

$$\Phi_{CR} = \frac{W_{CR}}{A_{dep}} = \frac{\left(\frac{16\pi\sigma^3}{3(P_v - P_L)^2}\right)}{\pi\left(\frac{M_L P_v}{\rho_L RT}\right)^{2/3}\left(\frac{2\sigma}{P_v - P_L}\right)^2} \tag{51}$$

$$\Phi_{CR} = \frac{\left(\frac{16\sigma^3}{3(P_v - P_L)^2}\right)}{\left(\frac{M_L P_v}{\rho_L RT}\right)^{2/3}\left(\frac{2\sigma}{P_v - P_L}\right)^2} = \left(\frac{\rho_L RT}{M_L P_v}\right)^{2/3}\left(\frac{4\sigma}{3}\right) \tag{52}$$

$$\Phi_{bubble} = \frac{E_{bubble}}{A_{dep}} = \frac{\frac{16\pi\sigma^3}{3(P_v - P_L)^3}\left[P_v\left(3 + \frac{h_{vap}}{RT}\right) - P_L\right]}{\pi\left(\frac{M_L P_v}{\rho_L RT}\right)^{2/3}\left(\frac{2\sigma}{P_v - P_L}\right)^2} \tag{53}$$

$$\Phi_{bubble} = \left(\frac{\rho_L RT}{M_L P_v}\right)^{2/3}\left(\frac{4\sigma}{3}\right)\left[\frac{P_v\left(3 + \frac{h_{vap}}{RT}\right) - P_L}{P_v - P_L}\right] \tag{54}$$

$$\Phi = \frac{P_{Laser} t_{Pulse}}{A_{focus}} \tag{55}$$

It should be noted that M_L is the molar mass of the liquid. The equations make the assumption that the bubble will form at the laser focus point, that the entirety of the laser fluence is available for bubble formation at that point, and that the liquid conditions are sufficiently far from the spinodal limits that their influence on the energy requirement is small. They also need the laser focus area to be greater than the bubble's energy deposition area. In addition, the pressure in the liquid needs to be below the liquid's vapor pressure.

This adds up to a constant value in Eq. (52); it followed the use of the critical energy, Eq. (19), from nucleation theory. A separate treatment uses the total energy for

bubble formation from Eq. (25), resulting in Eq. (54). In the limit as the liquid pressure gets arbitrarily negative, it asymptotically approaches the value given in Eq. (52).

It also ignores the effects of irreversible processes, thermal conductivity in the fluid (which can be significant in the time and length scales under consideration), and any sort of activation energy necessary for the event. It also assumes that incident radiant energy would be converted to ordinary heat, but this may not be the case. The effects of thermal conductivity can be limited by using a sufficiently short pulse period, if the laser is sufficiently powerful to emit the requisite energy in such a short period.

As for the activation energy, it may be possible to define a limit on its value from examination of relevant fluid properties. If the necessary activation energy is greater than the energy necessary to grow a bubble to the critical size, then delivery of the full bubble growth energy may not be sufficient to nucleate the bubble; the necessary energy would then depend on delivering the full amount of activation energy. It may be possible to place a limit on the activation energy by determining how much energy it would take to heat the volume of liquid under consideration (that necessary to fill a bubble of the critical radius at the vapor pressure of the bulk liquid) to the liquid spinodal limit at the operating CTMFD centerline pressure. Past the spinodal limit, nucleation is assured. If the spinodal limit at the operating pressure is unknown, one may use the liquid's critical temperature instead as a conservative limit; the spinodal limits do not exist past the critical point [6]. At the critical point, each molecule has enough energy to separate from all the other molecules, and the surface tension drops to zero [6]; the distinction between the liquid and vapor fluid phases disappears. This

estimate is shown in Eq. (58); if it is greater than Eq. (54), then the process may be dominated by the activation energy, but this is not expected to be the case. The estimated activation energy is expected to be a severe overestimate.

$$E_{act} < m_{condensed} c_p \left(T_{\lim} - T\right) \tag{56}$$

$$E_{act} < \frac{4\pi}{3} \rho_L r_{condensed}^3 c_p \left(T_{\lim} - T\right) \tag{57}$$

$$E_{act} < \frac{4\pi}{3} c_p \left(T_{\lim} - T\right) \frac{M_L P_v}{RT} \left(\frac{2\sigma}{P_v - P_L} \right)^3 \tag{58}$$

As an example, the values for acetone at 25° C are given below. The properties for acetone are listed in Table 1 [21] and Table 2 [22]. The data in Table 2 is used in the Antoine Equation [22] to determine the vapor pressure at a given temperature; the Antoine Equation is given in Eq. (59). With those parameters, the vapor pressure of acetone at 25° C would be 30.6 kPa. The resulting plots of the necessary bubble formation energy and the laser fluence are given in Figure 6 and Figure 7. While in the plots, the activation energy dominates for pressures above about -6 bar, the activation energy term is meant to be little more than an upper bound rather than a best estimate.

Table 1: Properties of Acetone [21]

Boiling Point	56.05° C
Surface Tension	0.02272 N/m
Thermal Conductivity	0.161 W/m-K
Density	784.5 kg/m^3
Viscosity	3.06E-4 Pa-s
Heat Capacity	126.3 J/mol-K
Latent Heat of Fusion	29.1 kJ/mol at 56.05° C
	30.99 kJ/mol at 25° C
Molar Mass	56.08 g/mol
Critical Temperature	508.1 K
Critical Pressure	4.7 MPa

$$\log_{10}(P) = A - \frac{B}{T + C} \tag{59}$$

Equation (59) uses the parameters given in Table 2 to determine the vapor pressure of acetone within the range of the given parameters. This can be used, along with the properties in Table 1 and the energy relationships in Eqs.(19), (25), (49), (50), (52), (54), and (58), to produce the curves shown in Figure 6 and Figure 7.

Table 2: Antoine Parameters for Acetone [22]

[P]	bar
A	4.42488
B	1312.253
C	-32.445
Range	259.16 K to 507.6 K

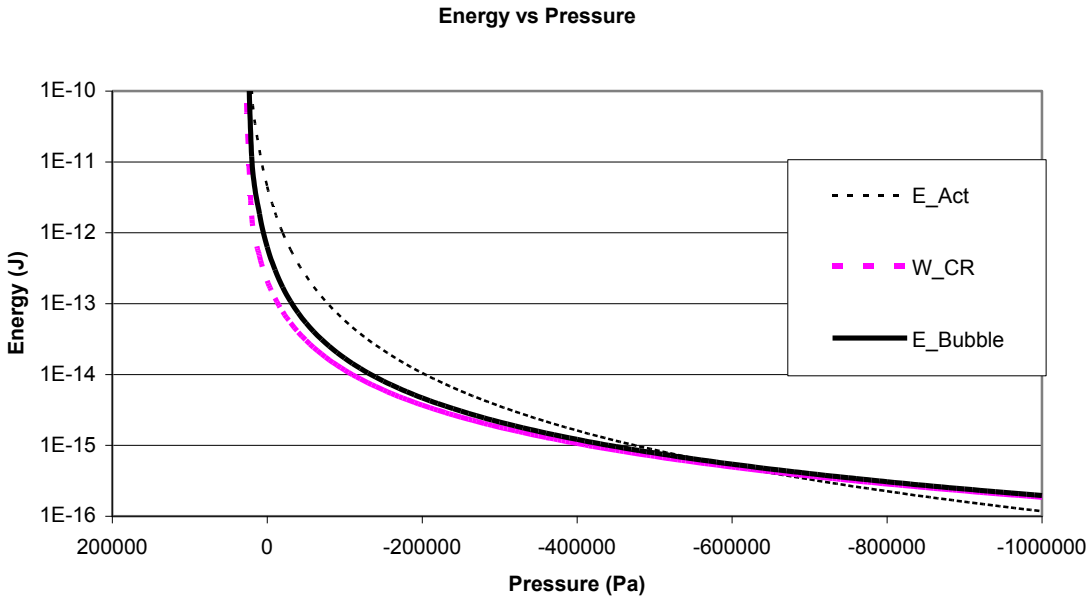

Figure 6: Energy Requirements for Bubble Formation

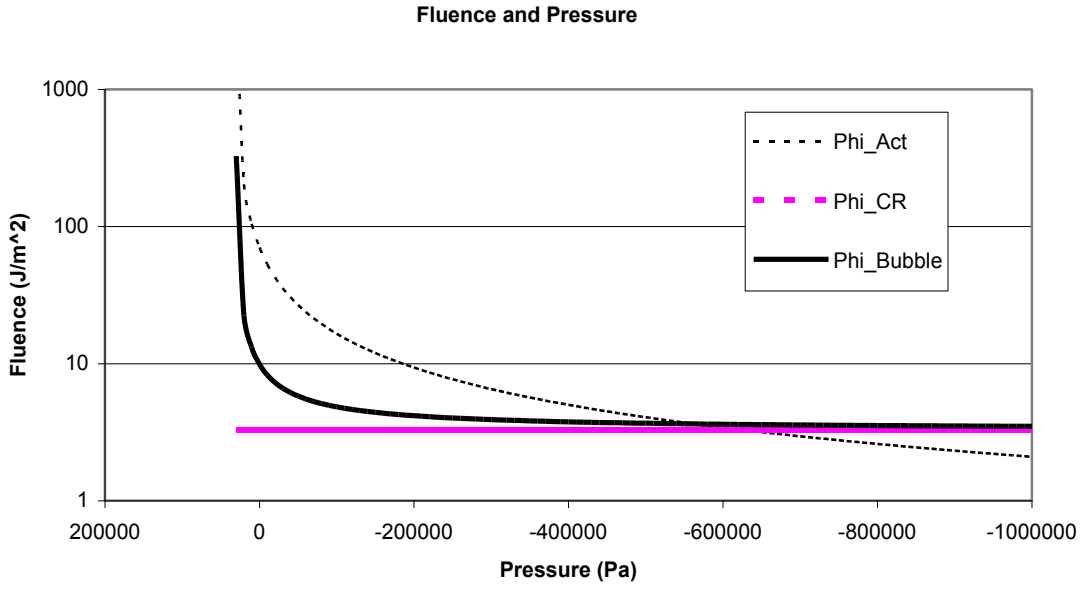

Figure 7: Fluence Requirements for Bubble Formation

Unfortunately, things are rarely as simple as the calculations behind Figure 6 and Figure 7. In this experiment, the working fluid was not a single, homogenous liquid. In order to ease the energy requirements for successfully inducing cavitation and forming a vapor column, solid 'seed' particles were mixed in with the liquid. These provided two benefits: weak points in the liquid to reduce the necessary localized energy deposition, and a much greater absorption of energy to help localize additional energy deposition from a given laser pulse. They are widely dispersed so as to limit the extra absorption in the beam path before the focus point, but are intended to be close enough together to assure that seed particles will be present in the region of high energy intensity within the depth of focus. While these may be true, their presence introduces new phenomena and additional complexity.

The particles chosen were black toner particles from a Dell™ Color Laser Printer 3110cn. If they are assumed to be made entirely of carbon black, they would have a density of 1.887 g/cm^3 [23], which is significantly more than that of acetone. Even when they would remain in suspension under ordinary conditions, it is possible that they may not when exposed to sufficient operational periods in a CTMFD. They may come out of suspension as a result of severe centrifugation. For example, at 200 rps and 5 mm out from the centerline, the centripetal acceleration is more than 7,895 m/s^2, which is nearly 805 times the acceleration due to gravity at the surface of the Earth. In addition, they do not have a uniform size, but have a potentially wide size distribution. They also introduce an added thermal sink at the point of greatest laser energy absorption; the energy directed to heating the toner particles will reduce the amount going to heating the

acetone. The particles, under the immense sudden heat load, may catastrophically deform, break apart, melt, or undergo other interesting thermochemical changes. This can lead to the formation of vapor columns that are less directly connected to the properties and energetics of the bulk acetone and have more to do with phenomena primarily occurring in the seed particles themselves. At present, it is unknown what kind of effects to expect.

A discussion of the theory behind the CTMFD experiment would be incomplete without mentioning some of the theoretical underpinnings behind the support equipment. This includes Pulse Width Modulation, RS-232, and some relevant optical theory.

3.1.1. Pulse Width Modulation

There are situations where it is useful to represent analog signals in digital circuits. This can be achieved by sampling the signal at regular intervals and encoding it into a more suitable form, making sure that the sampling frequency is high enough to capture all the important variations in the original signal. One common encoding technique is Pulse Width Modulation, which "represents a signal by using pulses of constant amplitude but variable widths" [24]. A simple PWM signal operates on a fixed frequency; as a result, the meaningful width of each pulse is limited to a maximum of the period of the PWM frequency. Therefore, a simple PWM signal is a square wave where the widths of the peaks are varied in relation to the encoded information. The fraction of time in which the signal is high rather than low is known as its duty cycle [25], and has a simple relationship with the original signal. If, for example, the original signal was at 75% of its limit, the corresponding duty cycle of the PWM signal would also be 75%. If

the original signal dropped to 25% of its limit, the corresponding duty cycle of the PWM signal would drop to 25% as well.

When a PWM is passed through a low-pass filter, the output is a reconstruction of the original signal that was encoded with PWM. So long as the original signal does not get too close to its upper or lower limits, the reconstructed signal will match the original signal [24].

PWM is commonly used in power conversion circuits; the digital logic helps enhance the efficiency of those circuits [25]. It can also be expanded beyond simple bi-state logic to three or more levels, depending on the application; for example, a three-level PWM signal might have voltage states of +1, 0, and -1 V [24].

PWM variants can be used in circuits that are not strictly digital. For example, an ordinary thyristor-based dimmer switch [26] typically trims the early part of each half-cycle in an AC line; the 'pulse' is the remaining part of the sine wave in the latter part of each half-cycle downstream of the dimmer switch.

3.1.2. RS-232

RS-232 ("Recommended Standard 232") is a standard that defines a type of serial communication interface commonly found on computers and associated equipment. It defines what has become the traditional 'serial port' found on PC-compatible computers since the original IBM PC. The standard itself predates the IBM PC. It is a robust interface that can tolerate abuse, and is relatively simple to interface with in software. In addition, user-mode software in Microsoft ® Windows™ operating systems are given a

remarkable degree of control over the hardware itself; direct software control of the 'control lines' is permitted. This makes it an attractive interface for a variety of uses.

In RS-232 standards, there are two device types: Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) [27]. The DTE would typically be a computer or other terminal, while the DCE would be a modem or printer connected to the computer's serial port; the DTE connects to the DCE in standard setups. The standards also define a full set of 25 pins, however, it is common for many of the pins to be absent from an interface. There are recommended uses for all the pins, but many of them are allowed to be used as seen fit by hardware and software developers.

In RS-232 systems using the DE-9 style interconnects (most PC serial ports are such), there are 8 signal lines and one ground line. The ground is a common signal ground for both sides. Instead of using a 'balanced' 4 input and 4 outputs lines, there are 3 lines signaling in one direction and 5 in the other. Which set is input and which is output then depends on the device type. This is not to be confused with 'unbalanced' electrical circuits; however, RS-232 is unbalanced electrically as well.

The DTE has 3 signals as its output: Data Terminal Ready (DTR), Request To Send (RTS), and Transmit Data (TxD or TD). Those signals are input signals on the DCE. The DCE, then, has 5 output signals: Data Carrier Detect (DCD), Data Set Ready (DSR), Ring Indicator (RI), Clear To Send (CTS), and Received Data (RxD or RD) [27]. Apart from the data lines TxD and RxD, the other 'control lines' have meanings or operations that are largely defined by software; user-mode software can easily take direct control of the state of the control lines.

The RS-232 standards also define voltages for the interface. Compliant devices are required to be able to have any of their pins shorted to each other or to ground without damage. The signal lines are also required to be able to handle having +/- 25 V applied, again without damage. However, the signals are required to be between +/- 5 V and +/- 15 V at the source; the receiving end allows the minimum to drop to +/- 3 V.

The region between -3 V and +3 V is undefined. Valid signals are therefore between -15 V and -3 V, and between +3 V and +15 V with respect to ground. For most of the signals, positive voltages reflect a logical 1, and in RS-232 documentation this state may also be called "true," "on," "asserted," or "mark." Negative signals, then, reflect a logical 0. Other terms for that state include "false," "off," "not asserted," and "space." The control lines follow such logic, but the data (TxD and RxD) do not; the data lines use inverted logic, where a negative voltage reflects a logical 1 and a positive voltage reflects a logical 0. The data lines also do not allow for the type of user-defined functionality that the control lines do.

Valid signals are not allowed to linger in the -3 V to +3 V zone; they are only allowed to be there when transitioning from one state to another. This wide voltage gap between defined states gives a degree of noise immunity to the system

The RS-232 standard also define limits on such characteristics as the rate of change for signal voltages (the slew rate) and the maximum data rate. However, it is common for these limits to be ignored in modern systems in order to allow for faster communication speeds.

3.1.3. Optics

In many materials, certain wavelengths of light can travel relatively unimpeded, while others may be absorbed in the material. It may also experience various types of scattering, and can reflect off of the material's interfaces. Some wavelengths may even cause the material to fluoresce. In the case of simple absorption, quantification of the absorption is simple and straightforward. The fractional change in intensity of light over a distance through the material depends on the material's properties and the distance traversed. This relationship is known as the Beer-Lambert Law, and the absorption parameter is known as the Beer-Lambert coefficient [28]. The equations are given in Eqs. (60) and (61). Attention should be paid to the symbols in use in these equations. While I represents the intensity and N the density, σ is NOT the surface tension; instead, it represents a cross section for interaction in Eqs. (60) and (61).

$$dI = -\sigma N I dx \tag{60}$$

$$\frac{I}{I_0} = e^{-\sigma N x} \tag{61}$$

If the primary interest is in energy absorption around a specified depth, that is relatively easy to compute. For example, if the region of interest is between x and $x + y$ in some medium, then the fractional absorption in that region can be given by Eq. (62).

$$f_{abs} = \frac{I_x - I_{x+y}}{I_0} = e^{-\sigma N x}\left(1 - e^{-\sigma N y}\right) \tag{62}$$

This works for a homogenous absorbing medium, and may be applied to Eq. (55) or any of the other fluence equations. In those cases, the distance y would be on the order of the diameter of the sphere of to-be-vaporized liquid. If there are discontinuities,

things are a bit different. If the medium is largely transparent but has highly absorbing bits scattered throughout it, then most of the energy may be deposited at the absorbing sites rather than in the bulk medium. If the medium is transparent enough, and the absorbing sites have great enough absorption parameters, then the incident intensity on an individual site will be approximately the initial intensity, and it may absorb roughly all the incident energy. Therefore, the local absorption fraction could approach 1.

The clearest way to think about the Beer-Lambert dropoff in intensity is to think of following a laser beam through a medium; the further along the beam, the weaker it gets. This is not entirely due to absorption, however; even a well-collimated beam will spread out with distance. Laser systems commonly emit a beam that can be described by how close it is to having a Gaussian profile. A good laser will have its beam be very close to one of the modes of a Gaussian profile. With a Gaussian profile, the major parameters used to describe a beam are its diameter and divergence (the angle at which the beam spreads out). Both of those can be affected by optics in the beam's path.

If a lens is placed in the path of a coherent Gaussian beam, simple ray optics calculations are insufficient to describe the beam's characteristics after the lens. While it would seem that a beam could be focused down to an infinitesimally small spot by using a quality convex lens with a good laser, this is not the case. First, the absolute minimum spot size is determined by the wavelength of the light (λ). In addition, a Gaussian beam will not focus to a point; there is a minimum spot diameter (ω_0) determined by the beam's diameter immediately before entering the lens (D) and the divergence of the

beam after the lens, which is related to the focal length of the lens (F). This relationship is given in Eq. (63) [29].

$$2\omega_0 = \left(\frac{4\lambda}{\pi}\right)\left(\frac{F}{D}\right) \tag{63}$$

$$DOF = \left(\frac{8\lambda}{\pi}\right)\left(\frac{F}{D}\right)^2 \tag{64}$$

The region in which the beam is close to its minimum size at the focus point has a finite and easily determined size. This so-called depth of focus, given in Eq. (64) [29], is the range centered around the beam waist at which point the beam width is SQRT(2) times as wide as the beam waist. It should be noted that Eqs. (63) and (64) are approximations of the Gaussian parameters that work for relatively small focal lengths.

3.2. EXPERIMENTAL EQUIPMENT

The experimental setup was constructed in the Nuclear Heat Transfer Systems Laboratory at the Riverside annex to Texas A&M University. Walls were constructed in the lab to create a room for the experiment, and the room was provided with an appropriate interlock system for the Class IIIb laser used in the experiment. A photograph of the new room can be seen in Figure 8. Like the walls, most of the equipment used in the experiment were custom-built.

Schematics for all of the custom electrical equipment built for this research are given in Appendix A. Similarly, the source code for the SpeedControl application developed to operate the facility is given in Appendix B.

Figure 8: New Laser Room

The experiment has a small glass CTMFD test section assembly attached to a rotating shaft, which is connected to a rotary tool; the shaft and CTMFD were custom-built, while the rotary tool is a standard power tool available at hardware stores. They in turn are mounted to an off-the-shelf Dremel ® stand using scrap wood and hose clamps. The system is instrumented with a custom-built IR speed sensor and optical cavitation sensor, while the speed is controlled by connecting the rotary tool to a custom-built controller. Much of this can be seen in Figure 9.

Figure 9: CTMFD Experimental Facility

In Figure 9, most of the key components in the experiment can be seen. On the left, somewhat cut off, are the computer system and oscilloscope. On the same platform, one can see the RS-232 isolator (bottom), the cavitation sensor box (sitting on the RS-232 isolator), and the pulse generator (nondescript box next to the oscilloscope with various wires). Mounted on the containment box are the box interlock (middle), speed controller electronics (top), laser interlock emergency stop button (red button on the right), and the experiment (inside). The high speed camera is visible on the bottom, as is the optical platform. The laser is hidden from view; it can be seen in the bottom in Figure 10.

Figure 10: CTMFD Experiment with Open Enclosure

There are optics that focus the laser (an off-the-shelf system) to a point within the CTMFD test section. A close-up showing the relation between the focusing lens, test section and cavitation sensor is shown in Figure 11. While some of these electrical and optical components are outside, the mechanical components are housed in a heavy containment box. The containment box is interlocked with custom electronics for safety reasons, and is separated from the laser and optical mounts to reduce the transmission of vibrations to the optical assemblies.

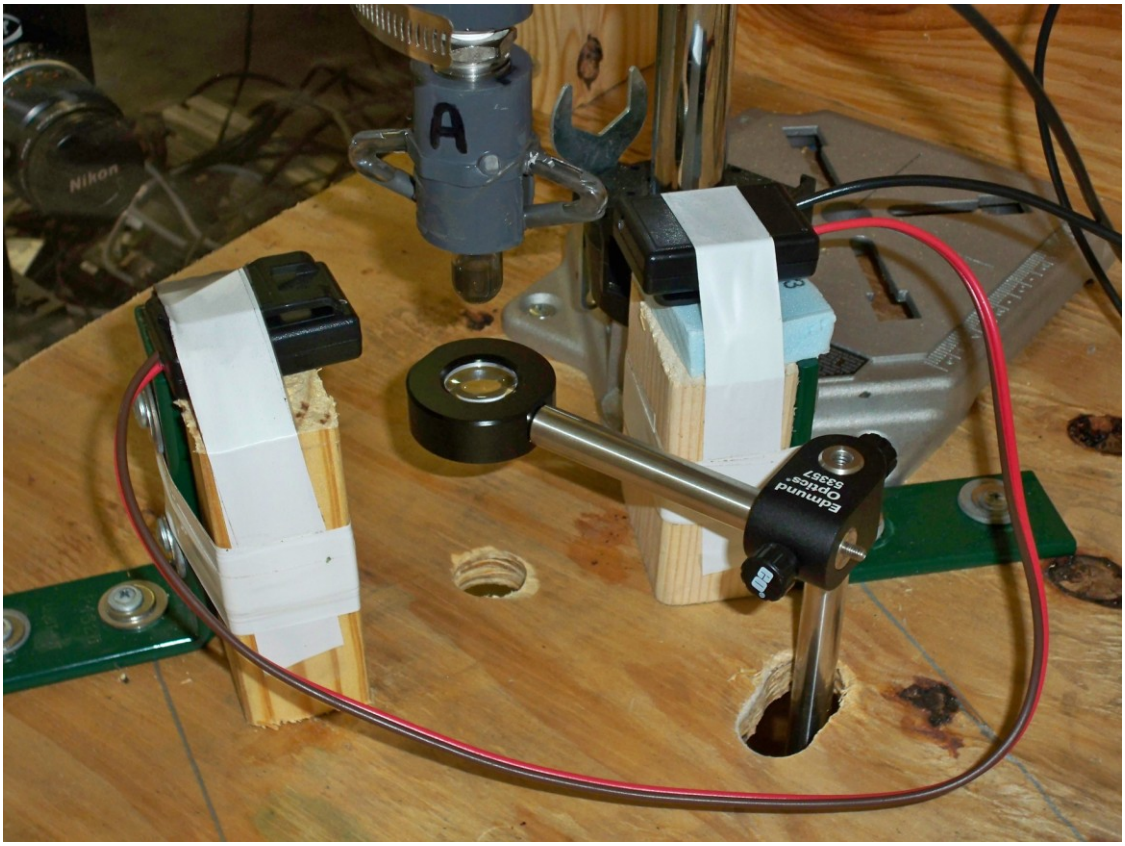

Figure 11: Close-Up of the Mounted Test Section

The speed sensor, cavitation sensor, speed controller, and laser head are connected to a custom central power supply and RS-232 interface through applicable adapters. In addition, the laser head interfaces with the room interlock system and the pulse generator circuit, both of which were custom-built. The pulse generator circuit, in turn, connects to the RS-232 interface as well as an oscilloscope. The oscilloscope and RS-232 interface connect to a desktop computer system for data acquisition and control. While the computer and its operating system are off-the-shelf commodities, the data acquisition and control software had to be written from scratch in a major development effort.

RS-232 serial communication was used in the experiment, but not all signals were used as one might expect from standard RS-232 systems. However, an RS-232 serial port on PC hardware is flexible; in addition to the standard text transmission and simple logic states, loosely-applied pulse width modulation was used (see Sections 3.1.2 and 3.1.1). Details for each of the components involved in the experiment can be found in Sections 3.2.1 through 3.2.16.

The above setup works well when calibrated. However, there are phenomena that it cannot monitor that still garner interest. To gain insight into the dynamics of the formation of the vapor column, a high speed camera was employed. However, due to operational difficulties, it is not a standard instrument in the experiment. It was only used in dedicated runs to collect high speed video of the cavitation events and the rapid growth of the bubbles into vapor columns.

3.2.1. Containment Box

The containment box is an important part of the experiment. Not only does it provide physical support to its contents, it also serves as a major component in the experiment's safety systems. It is more than just a large, heavy assembly of wood, plastic, and metal.

Physically, the containment box is built up from a lower 2' x 4' platform made of plywood and 2x4 boards. From the platform, two similarly-sized regions are formed: the experimental area, and a secondary area. Both are rectangular volumes sitting on the platform with 2x4 framing. It rises about 3' above the platform. The experimental area

is separated from the secondary area by a plywood wall. The remaining four sides of the experimental area are covered by 0.25" polycarbonate windows.

The thick polycarbonate was chosen to ensure safety in the event of catastrophic failure in the experimental area. As the rotary tool is capable of speeds in excess of 35,000 rpm (>583 rotations/sec), ensuring safety became a central concern. At that speed, a device attached to the rotary tool extending out to a radius of just 5 cm would have a tangential velocity greater than 183 m/s at the edge. While it was expected that such speeds would not be readily attained, even much slower speeds could still pose a serious threat. The 0.25" of polycarbonate was deemed necessary to prevent the escape of projectiles from a failed test section. Should a catastrophic failure occur, the plywood and polycarbonate enclosure will keep the operator from catastrophically merging with the failed test section.

While providing physical protection, polycarbonate also provides clear, scratch-resistant window for observing the experiment in progress. Since the front, top, and left and right sides have the large polycarbonate windows, there is a large choice of angles to view the experiment from.

In addition to the windows, the front and top are hinged such that they can be rotated up as a unit and flipped over the top of the secondary area; it swings open. This provides a free and clear opening to the experimental area, with unobstructed access from the front and above. The side windows remain in place. When the access is closed, a lever arm can be rotated into place that will lock the front and top in the closed

position. The interlock on the box senses the position of both the access and the locking arm, so that attempts to open the box will result in power being cut from the experiment.

The interlock is mounted to the box, on the wall in the secondary area. Aside from the frame and the floor and wall to the experimental area, the secondary area is open to the air. In addition to the interlock for the box, the speed control unit is also mounted to the wall. Two switches are mounted between the interlock and speed control units; these turn on and off the outlets in the experimental area. One switch turns the regulated power (the speed control line) on and off, while the other switch controls raw 120 VAC in the experimental area for accessories such as lights.

The wires for power and sensors are fed through penetrations in the plywood wall in the back of the experimental area into gang boxes on the inside. The AC power lines (both regulated and unregulated) are attached to outlets in the experimental area to supply power to the rotary tool and the experimental area's lights. The wires for the speed sensor and cavitation sensor heads go through a separate penetration into a gang box with a cover in place. They then go through small openings in the box to reach their respective pieces of equipment.

In addition to the interior electrical boxes, there is a horizontal support board near the top of the experimental area. This holds a fluorescent light that illuminates the interior of the experimental area; the light plugs into the unregulated AC socket. The support board also allows the Dremel ® stand to be secured at its top, restricting wobble when the experiment is running. This has the side effect of making the experiment louder and transmitting additional vibration to the containment box from the experiment.

The floor of the experimental area has two major penetrations; both can be seen in Figure 11. One allows the optical support rod to enter the area and support the lens without being attached to the vibration-laden containment box. The second penetration is directly below where the test section is mounted to the shaft; this allows the laser beam to enter the experimental area and be focused into the test section. It is a relatively large and unobstructed penetration, but it is positioned such that it is extremely unlikely for shrapnel from a catastrophically failed experiment to traverse, bounce off something on the other side, and cause injury to persons in the vicinity.

The entire containment box is mounted on feet, one at each of its four corners. This elevates the system several inches above the ground. Each of the feet has a thick rubber strip glued to its bottom; this is intended to reduce the amount of vibration that can be transmitted to the floor beneath the containment box. There is sufficient room underneath the containment box to provide space for the laser head and optical assemblies.

3.2.2. Containment Box Interlock

The interlock on the experiment's containment box was the first circuit to be designed and constructed. It was recognized early on that hazardous conditions would exist near the operating experiment, and that some physical separation between experimenter and experiment would be a prudent measure in order to keep the two from becoming one. While the box itself works when procedures are strictly followed, it is a good practice to include the interlock to help enforce safe practices. Should the box be opened, the interlock acts as a 'kill switch' and cuts the power from the box, shutting

down the rotary tool and quickly slowing the rotation down to zero. It does not, however, affect the operation of the laser.

The containment box interlock centers around two relays. The detector relays is connected to two microswitches on the experiment: one on the door itself, and a second one in series with the first on the locking arm. If the door or locking arm is opened to give physical access to the experiment, the detection circuit is opened and the detector relay opens. In addition, the second (downstream power) relay opens, cutting off power to the experiment. This allows separation of the low and high voltage sides of the circuit and maintains the circuit status -- when the containment box is closed and latched, the interlock will not automatically reconnect power to the experiment.

The interlock circuit displays its current status via indicator LEDs, and requires manual 'arming' of the system. Two buttons serve that purpose: one to arm the system, and one to manually trip it. When the 'arming' button is pushed, the system will only arm if the containment box is closed and latched, and if the 'trip' button is not concurrently pushed. Otherwise, the interlock will not connect power to the downstream components, even while the 'arm' button is held down.

In addition to providing an interlock to the containment box, the interlock circuit can serve as a source of +12VDC power. However, it is not used with the current experimental setup.

3.2.3. CTMFD Hardware

The rotary tool is a Craftsman ® Rotary Tool Model 572610950. It has a maximum rotation rate of 35,000 rpm, and is rated to consume 1.15 Amps at 120VAC.

For the experiment, it plugs into the controlled socket inside the containment box, and has its switch set on high. It is mounted near the top of a fully-extended Dremel ® stand, which is mounted inside the containment box at its base on the platform and at the top where the vertical rod meets the support board. This can be seen in Figure 12.

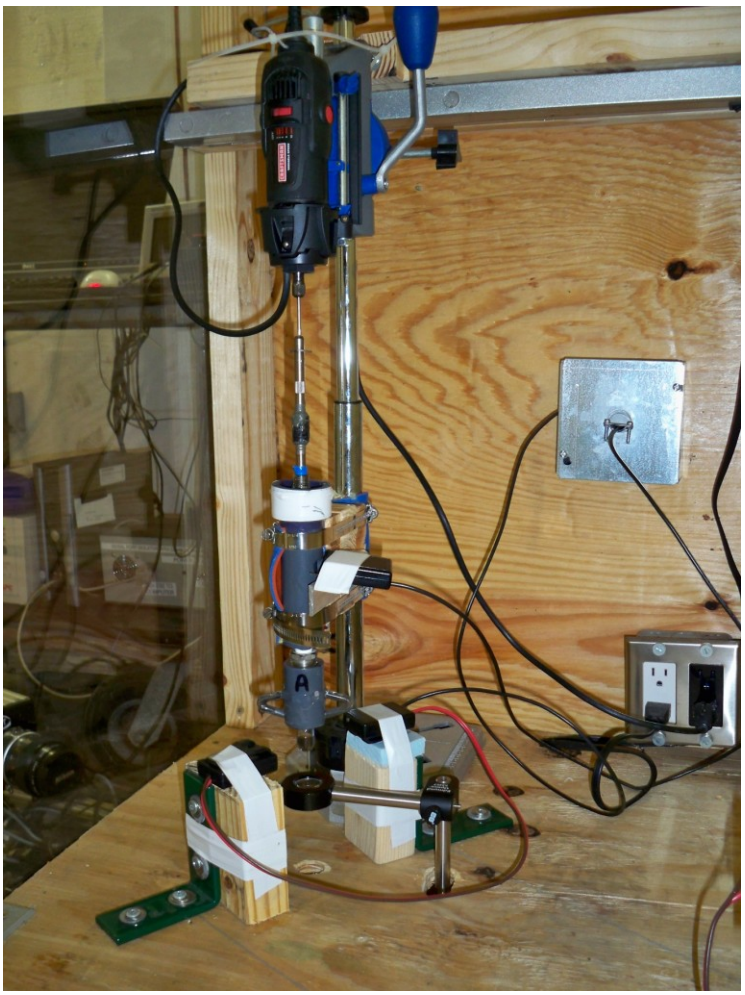

Figure 12: CTMFD Setup

The rotary tool attaches to a short 3 mm diameter by 60 mm length rod that has a pinhole at the far end. This rod is inserted into the receptacle for it in the main shaft, and the two are secured by passing a clip through their pinholes.

The main shaft itself is made up of several pieces. The top piece (7.96 mm diameter by 71 mm long) was cut from a paper feeding shaft in an older HP ® Deskjet™ 648C printer. A hole was drilled axially into the center of one end to insert the attaching rod for the rotary tool. A second, smaller hole was drilled through the side at the same end to allow a locking pin or clip to be inserted through the corresponding hole in the attaching rod when it is inserted and aligned. The opposite end has interlocking teeth cut into it to match a similar set on the lower portion.

The lower portion of the main shaft connects to the upper portion using sets of interlocking teeth cut into the mating surfaces, adhesive glue, and a plastic sleeve cut from paper feed wheels in the same inkjet printer that the main shaft came from (both parts were cut from the same rod). The sleeve covers the joint between the upper and lower portion, and everything is held together by the glue. It extends from the joint 36 mm down to the top of a pipe, traversing most of the length of the pipe.

Below the joint, the lower portion of the main shaft was wrapped in an even layering of aluminum tape and inserted into the center of a 0.25" SCHD 40 SS304 pipe with a length of 150 mm. They were secured together with epoxy. This expanded main shaft is marked using aluminum and black electrical tape running axially down the central portion; this allows for the speed sensor to see an alternating light and dark surface as the shaft rotates.

The expanded main shaft was then mounted to two high speed ball bearings, one near each end. The gap between the inner surface of each bearing and the outer surface of the pipe was filled with tape and epoxy for a permanent mount when the shaft was inserted into the outer housing.

The outer housing is made of 1" SCHD 80 PVC pipe cut to fit. Appropriate gaps were cut to hold the high speed bearings in place, and scrap PVC was used to make a mount at the top for clips to secure the upper high speed bearing in place. A hole was cut in the center of the housing for the speed sensor to view the rotating shaft inside.

The housing and shaft assembly was mounted vertically to the Dremel ® stand and held in the correct position using wood blocks, rubber hose used as a cushion/gasket, and worm gear hose clamps. A piece of aluminum was cut and mounted to the wood support blocks and is used to support the speed sensor; it is taped into place aimed into the hole in the shaft housing.

At the bottom of the shaft, an SS304 pipe bushing for 0.25" to 0.5" SCHD 80 was securely threaded onto the 0.25" pipe. The bushing is the interface between the main shaft and the test section assemblies; the test section assemblies are replaceable and can quickly and easily be threaded onto the bushing.

3.2.4. CTMFD Test Sections

The test sections are made up of two primary components bonded together by epoxy: a mount and the CTMFD glassware. The mount is a 0.5" SCHD 80 PVC pipe coupler with two slots cut in one end on opposite sides. These slots allow the glass test section to be inserted into the mount while its arms extend out through the slots. Care is

taken to ensure an adequate alignment between the mount's center of rotation and the center of the bulb on the glassware, and the two components are then epoxied securely together. The mount threads onto the bushing on the bottom of the rotational shaft, allowing for quick and easy test section swaps. An assembled CTMFD test section can be seen in Figure 13.

Figure 13: Assembled Test Section A

The glassware is a small CTMFD made of 1 mm thick borosilicate. It has a bulb at the bottom, from which two opposing tubes extend out the top. The tubes extend out a distance, then have bends that angle them back toward the centerline of the bulb, still angled upward. They meet and rejoin each other a short distance above the top of the bulb, at which point they form a vertical tube. These arms, from where they leave the bulb to where they rejoin, form a diamond-shaped area between them. This can be clearly seen in Figure 14.

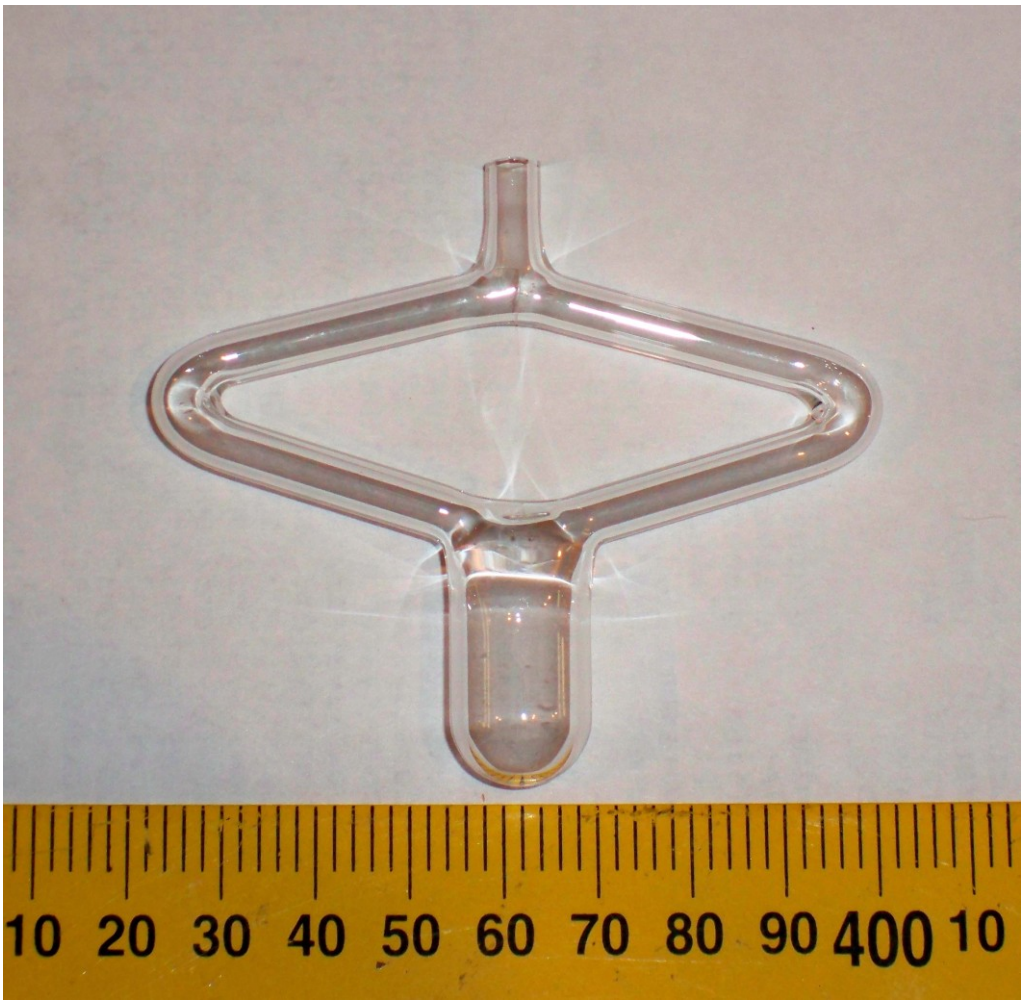

Figure 14: Test Section A Glassware

There are two major configurations for the glassware: a bulb with a hemispherical bottom, and a bulb with a flat bottom. The hemispherical-bottom bulbs match designs used elsewhere [5], while the flat bottom bulbs present a surface that should not have seriously unpredictable lensing effects. This should allow for better focusing of the laser beam to a point within the bulb than is possible with the highly curved hemispherically-bottomed bulbs. A flat-bottomed bulb can be seen in Figure 15.

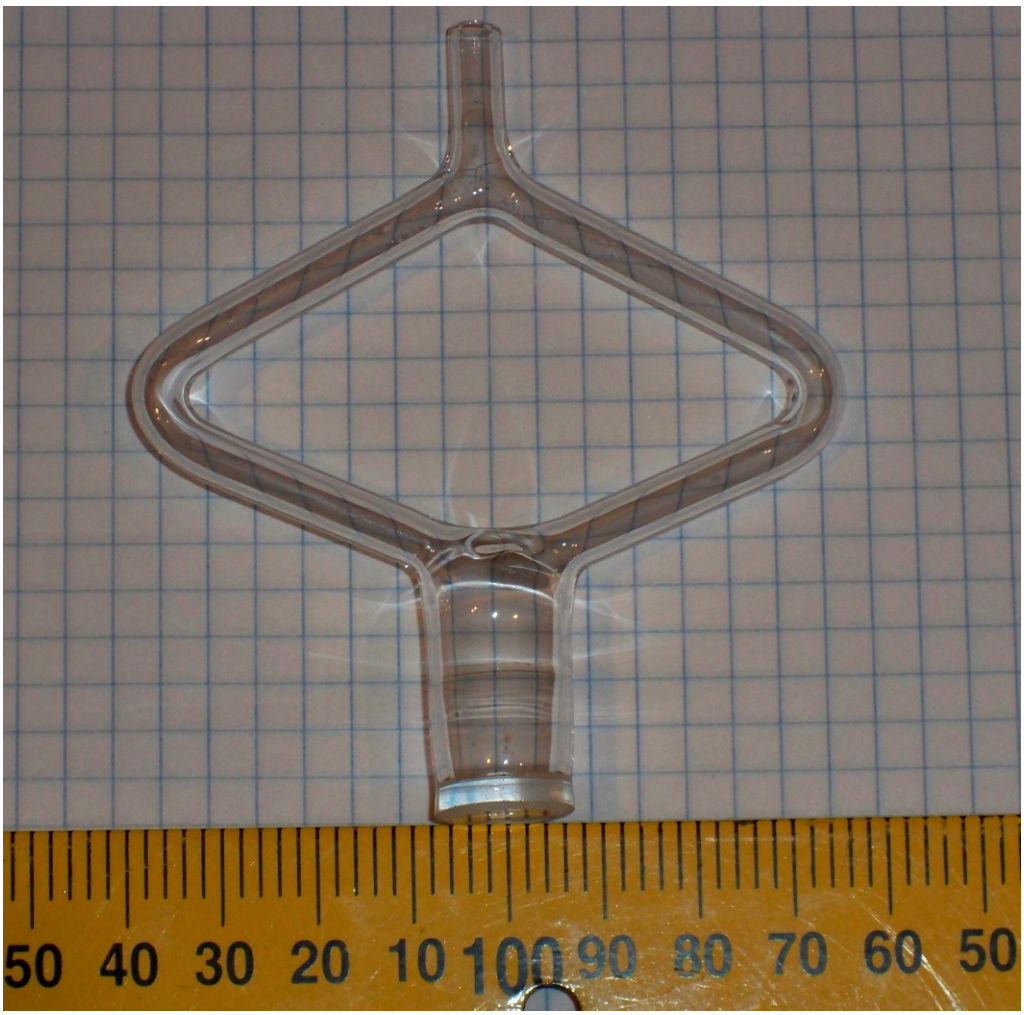

Figure 15: Flat-Bottomed Glassware

The hemispherically-bottomed bulbs have the following nominal dimensions: 15 mm diameter by ~30 mm total height in the bulb The inner width for the diamond-shaped region is between 60 and 65 mm, with an outer width between 74 and 80 mm. The inner height of that area is on the order of 18 to 20 mm. The flat-bottomed bulbs have the following dimensions: a cylinder of 15 mm in diameter and a height of 31 mm. The flat-bottomed bulbs were not used for the testing in this thesis.

3.2.5. Speed Sensor

The speed sensor is a relatively simple circuit; it is essentially an optical tachometer. It consists of three major parts: a sensor head, a control unit, and a signal adapter. Each part is separated and placed in its own enclosure with cables connecting the parts.

The sensor head consists of an enclosure mounted near the shaft of the experiment. The shaft is divided radially into two regions: a dark half and a reflective half. This was done by applying black electrical tape and aluminum tape along the shaft; when it rotates, it alternates between the black and reflective surfaces being presented to the sensor head. The sensor head itself has an infrared LED and an infrared photodiode. The IR LED illuminates the rotating shaft, while the IR photodiode is used to transform the amount of reflected radiation into a voltage. That voltage is fed back to the control unit.

The control unit feeds the voltage from the sensor head into an op-amp, which is used as a comparator. It differentiates between the light and dark regions on the shaft by comparing the sensor output voltage to a tunable reference voltage. This reference

voltage is developed as part of the calibration of the sensor. To aid in this, two visible LEDs are included and connected to the output signal: a red LED indicating a low state, and a green LED indicating a high state. The reference voltage itself comes from a potentiometer connected to a knob. This is adjusted to determine the threshold between high and low output states for both the reflective and dark regions on the shaft; it is then positioned midway between the two limits to provide maximum noise immunity on either side. When properly operating, the control unit outputs a square-wave signal at the frequency of rotation. A high output state corresponds to the reflective side of the shaft being in view of the sensor, while the low state corresponds to the dark side of the shaft. At constant speeds, the times spent in the high state and low state should be similar.

The third enclosure, the signal adapter, was a later modification to the circuit. As the speed sensor was designed and constructed early on, before the DAQ design was fully worked out, it resulted in an output signal that was not compatible with the final system. The signal adapter takes the square wave from the control unit and changes the signal voltage levels to those compatible with RS-232 systems. With the adapter in place, the signal (still a square wave at the frequency of rotation) can be used directly in RS-232 control lines.

3.2.6. Cavitation Sensor

The cavitation sensor was designed to be very similar to the speed sensor. Originally, it used an almost identical circuit based around an IR LED and photodiode. There were only two differences: it did not need a signal adapter, and the sensor unit

was split into two units. The IR LED was in one small enclosure, and the photodiode was in the other small enclosure. This was intended to allow for flexibility in placing the units as well as the ability to pick up direct rather than reflected IR radiation from the IR LED.

However, the original design proved difficult to get working in practice. As a result, some minor modifications were made. The IR LED and IR photodiode were replaced with visible-spectrum optoelectronic components; a red LED was substituted for the original IR LED, and a CdS photocell replaced the original IR photodiode. While CdS photocells have much slower response times than photodiodes generally do, that is not a problem with the current experimental setup.

The cavitation sensor works by illuminating the bulb in the test section with the red LED. When it is filled with a largely transparent fluid, certain lensing effects will come into play. When there is no vapor column in the bulb, only liquid, the bulb will tend to concentrate the light from the LED on the opposite side of the bulb. This bright spot is where the photocell is placed. When cavitation occurs and a vapor column forms, the concentration of light on the photocell is reduced. If the vapor column is wide enough, almost all of the light originally concentrated on the photocell will be scattered elsewhere, and the photocell will see darkness.

The cavitation sensor is mounted on wood blocks secured to the lower platform inside the containment box. They are aligned and aimed at each other, with the bulb of the test section in between them.

The cavitation sensor is calibrated in essentially the same manner as the speed sensor. It has a knob and indicator LEDs on the control unit that function the same way they do in the speed sensor. The knob is turned until the thresholds are found for both the case where a vapor column exists and the case where it does not. The knob is then adjusted to a point between the two thresholds. Alignment beforehand is necessary. It should be noted that additional care must be taken with the cavitation sensor. Not only does the setpoint have a tendency to drift, but it also has a high degree of sensitivity to interference from light sources. Ambient illumination changes the setpoint, and care must be taken that changes in the illumination levels are not enough to impair operation of the sensor during testing. Sufficient brightness may even prevent the sensor from operating at all, regardless of the setpoint.

It should be noted that the signal that the cavitation sensor outputs is inverted; when the design was changed, the meanings of 'high' and 'low' output states were swapped. As a result, a separate signal inverter was assembled and connected to the output of the cavitation sensor.

3.2.7. Signal Inverter

The signal inverter is a very simple circuit centered around an op-amp used as a comparator. It was built as a result of the way the cavitation sensor operates, but it is an independent device that can be easily connected and disconnected from any of the binary state signals used in the experiment.

The signal inverter takes an RS-232 signal and reverses its polarity. Therefore, if the input is 'high' then the output will be 'low,' and vice versa. It can be useful for

dealing with changes in the meanings of the high and low states that are not accounted for elsewhere in the setup. It is currently used to address the change in the meaning of the output signal from the cavitation sensor.

3.2.8. Speed Controller Electronics Unit

The speed controller unit has two largely separate functional circuits. The first circuit is a protection system designed to detect faults in the power line. It will disconnect the hot, neutral, and ground lines in the event of a polarity reversal or other wiring fault in the electric source. As such faults have been discovered in the wiring at Riverside in the past, this protection was determined to be warranted. Since the upstream interlock chassis is metal, its ground was connected to the protection circuit as well to prevent shocks in the case of a hot-ground reversal.

The second circuit uses a form of pulse width modulation to control the speed of the attached rotary tool. This output is the same as that of a conventional dimmer switch. As the AC voltage and current cross zero, the silicon controlled rectifiers in the circuit return to their non-conductive state. Then, after a period determined by the timer and control signal, a signal is applied to the silicon controlled rectifiers and they transition to a fully conductive state. Even when the signal is removed, they remain in the conductive state until the current through them falls below a threshold value. The signal, however, is maintained until a short period before the incoming AC voltage crosses zero.

The product is a modified AC wave where the early parts of both the positive and negative cycle are cut off and remain at zero. This is a characteristic waveform for SCR

and TRIAC circuits, and constitutes a low-frequency variant of pulse width modulation where each pulse is the 1/120 second period where the input voltage swings sinusoidally from zero to its peak (either positive or negative) and then back to zero. By removing parts of the waveform in this manner, the output power can be controlled. It works well for incandescent light bulbs, but not all circuits can accept it. CFLs that are not specially designed for dimmer switch operation, for example, will not tolerate it. However, by trial and error, it was found that the rotary tool employed for this experiment was compatible with such a method of power control. Controlling the input power for the rotary tool translates well into a speed control. An example of the waveform output by the controller electronics is given in Figure 16; it was set near the mid-level, so roughly the first half of the AC sinusoid half-cycle is dropped. It should be noted that the oscilloscope in the photograph is not the one used for laser pulse measurements.

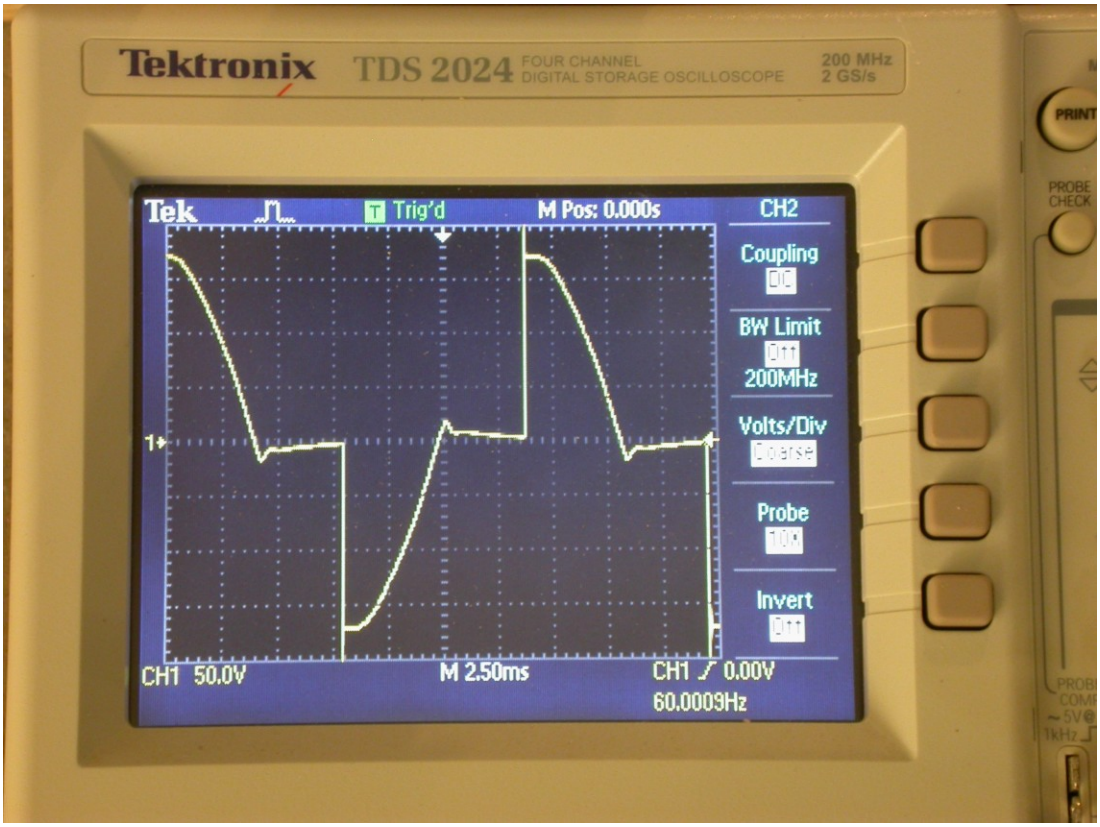

Figure 16: Controller Output Waveform

The speed controller has two options for managing the output levels: a control knob and an external input that accepts a modulated input signal at RS-232 voltages. There is a switch to select between the two. When the knob is used for manual control, it presents a control voltage from the position of the knob's potentiometer to an op-amp used as a comparator. The other signal comes from the timing circuit. The timing circuit uses a rectifier bridge and Zener-based voltage reference along with an op-amp to determine when the incoming AC voltage is near zero. When the voltage is not near zero, the op-amp sends a signal to a series of transistors that will charge a capacitor to reflect the AC phase angle. When the AC voltage swings back to zero, the op-amp (used

as a simple comparator) switches from a high output state to a low one, which triggers the transistors to quickly drain the timing capacitor. Therefore, the voltage across the capacitor is determined by the progression through each AC half-period.

When the voltage across the timing capacitor is greater than the reference voltage, another op-amp (again used as a comparator) sends a trigger signal to the driver circuit that fires the SCRs.

The second input option uses an external signal to generate a reference voltage to compare to the timing capacitor level. The external signal is connected to an op-amp, once again used as a simple comparator. The output from the op-amp will therefore be either high or low. This signal is fed into a simple RC-circuit that acts as a sort of low-pass filter. With a high-frequency input signal fed into the op-amp, the capacitor in the RC-circuit will trend toward an average value based on the time that the signal is high vs. the time that the signal is low. The higher the frequency, the lower the 'ripple' voltage across the capacitor will be and the closer the voltage will remain to the average. This can be thought of as a form of a digital-to-analog convertor; an input logic signal will be converted to an average value based on the time at either level, with a wide input frequency tolerance.

This wide input frequency tolerance is the result of the high RC time constant used in the circuit's design; the time constant is 2.2 seconds. The design called for such a high time constant to reduce the effects of poor timing control in the input signal; glitches and jitter will be readily filtered out. However, it does have the resulting effect of slowing down the response time to legitimate signal changes. The 'stroke time' for

changing from zero to full power is several seconds, which was deemed to be acceptable. Controllable speed fluctuations in the rotary tool were assumed to not need a faster response than was given by the speed control circuitry.

3.2.9. RS-232 Isolator and Power Supply

The main interface between the computer and the rest of the experiment is the RS-232 serial port isolator and +/- 15VDC power supply. It provides the necessary DC power to the sensors, signal inverter, and pulse generator. In addition, it provides connections to the RS-232 serial port on the computer.

The power supply for the isolator is a simple design. It uses a toroidal transformer to provide a reduced input voltage into the regulators and LC filtering to smooth out the output power. Rectifiers, inductors, and capacitors connected to the transformer provide positive and negative unregulated voltages with respect to ground as input to the regulators. The regulators are simple one-component linear regulators: an LM7812 on the positive rail and an LM7912 on the negative rail. Zener diodes and capacitors are used to adjust the ground reference voltage, giving +/- 15V regulated output. An additional set of filter capacitors are used immediately downstream of the regulators.

The serial port isolator circuit does not fully isolate the DTE from the DCE side [29]; the ground line is common to both sides and simple op-amp circuits are used rather than a fully independent optoisolator system. Four dual op-amp ICs are used, bringing the number of op-amps to 8, one for each signal line on the DE-9 connector. The heart of each signal isolator is identical.

On power-up, the default signal on each signal line is negative in the absence of an RS-232 signal. If an existing signal is removed after power-up, then the output of that line is maintained at the last input value (positive or negative). The op-amps are used as comparators, and positive feedback is used to help enforce a policy of either positive or negative output when the isolator has power.

According to RS-232 standards, the voltage range centered around ground - 3 V to + 3 V is a sort of 'dead zone' to help with noise immunity [27]. Signals are only allowed to exist in that range when changing states. Therefore, the isolator circuits were designed to maintain their current state until the input signal fully crosses that voltage region. At that point, it is clear that the input has changed states and the output will change to match it, aided by positive feedback.

For this experiment, the computer is treated rather conventionally as DTE, and the experiment's electronics are the DCE. Therefore, the isolators take 3 inputs from the computer side and output them to the experiment, while 5 inputs are taken from the experiment and directed to the computer. Not every signal line is used in the experiment; while all 3 outputs from the DTE are used, only 3 of the 5 inputs to the DTE are connected to the experiment. This leaves some room for additional connections in the future.

The computer is connected to the isolator with a standard 9-pin serial cable. The experiment then connects easily to the isolator using the banana jacks on the back. In addition to the 8 sets of banana jacks for the RS-232 signal lines (the signal line and a

ground connector), a 3-plug banana jack is there to conveniently supply +15VDC, ground, and -15VDC.

3.2.10. Laser System and Optical Assemblies

The laser system is a Coherent Cube 405-100C. It is a 100 mW Class IIIb diode laser with a nominal wavelength of 403 nm. It consists of the laser head, power adapter, heat sink and fan, keyswitch, and associated cabling.

The laser can connect to a computer via its USB port or through an RS-232 serial port; it only uses the TxD, RxD, and signal ground lines. Computer control over RS-232 is straightforward; it uses a documented command set that lends itself to easy automated control.

The laser also operates in several modes. In pure CW mode, the beam power can be controlled by issuing commands from a computer. It also offers two modes for a modulated beam: an analog power control option that allows modulation at up to 350 kHz, and an external TTL control for digital modulation up to 150 MHz. In digital pulse mode, the power is set externally and the beam is turned on or off based on the TTL state (high or low). Rise and fall times are less than 2 ns. The delay from signal to laser turn-on is 16 ns, and the fall delay is 15 ns [30].

The laser head is connected to the pulse generator through its SMB connector. It also connects to TxD, RxD, and a signal ground on the RS-232 isolator. In addition, it has two connections to the laser interlock system. The interlock loop is connected through the keyswitch. The interlock's flasher circuitry is connected to the appropriate terminals wired to the laser head. In addition to the interlock circuitry, the laser has a

shutter build into the laser head that will prevent laser radiation from escaping the laser head when closed.

The laser head is bolted down to the heat sink and fan assembly. Without the heat sink and fan, the laser will quickly overheat and may be damaged. The connected laser and cooler assembly is itself mounted to a stage that allows lateral translation, which greatly assists in beam alignment. The stage in turn is mounted to a platform made of plywood and 2x4s. The platform is bolted down to the lab floor with a thick rubber gasket between it and the floor to help resist the transmission of vibrations from the floor to the platform. The containment box is placed over and around this optical platform, but they are not attached to each other; the containment box's feet stand on rubber next to the platform, not on it. This was an attempt to keep as much vibration as possible from transmitting through the containment box to the optics.

Figure 17: Laser and Mirror on Platform

In addition to the laser assembly, an optical component assembly is mounted on a stage to the optical platform. This is shown in Figure 17. The component assembly consists of a vertically mounted support rod which rises into the containment box along with mirror and lens assemblies and their associated support arms. The mirror assembly consists of a first surface mirror on an adjustable mirror mount extended on a horizontal support arm that attaches to the vertical support arm with a connector. The mirror is positioned below the containment box laser penetration and angled to reflect the laser beam up through the penetration and into the test section. A plano-convex lens is positioned in the beam path to bring the beam to a focus within the test section; it is held

in a lens mount attached to a horizontal support rod which is mounted near the top of the vertical support rod. The lens is closely below the test section; this part of the assembly is inside the containment box.

3.2.11. Laser Interlock System

The laser interlock system is designed to address the laser safety requirements when used in conjunction with the corresponding circuitry built in to the laser system. It complements the built-in circuits in the laser head by providing the required warning lamps and interlock circuitry. When the interlock is tripped, the laser will stop firing.

The interlock system consists of the main interlock panel, the two warning lamps, the two emergency stop buttons, the door open sensors, and the associated interconnects. The circuit designs were based on schematics provided by Dr. Leslie A. Braby for an X-Ray system interlock.

The design uses relays to open and close the interlock loop based on the system status. On powering up, which requires a key, the system will not be armed. Both the inside and outside warning lamps will activate at half power. There are three separate interlock circuits in the panel. In order to operate the laser, all three must be in the 'armed' state. Two of the circuits are self-triggering; they only come into play if the warning lamps have burned out bulbs. If either the inside or outside warning lamp is burned out, the corresponding circuit will trip and an indicator LED will light up until the situation is rectified.

In order to arm the third circuit, the door must be closed and both emergency stop buttons must be pulled out to the 'ready' position. Then the 'arm' button on the

panel must be pushed. If the door is open or either emergency stop button is pushed, the circuit will not arm and will remain in the 'tripped' stat. If the power is turned off, the door is opened, or an emergency stop button is pushed, it will immediately enter the tripped state and the corresponding indicator LED will illuminate.

The laser system expects the external interlock to provide a short between the appropriate contacts to indicate that the interlock system is armed and ready; it is tripped when the connection is opened. Therefore, when any of the interlock circuits trip, their respective relays will open, breaking the connection and giving an open circuit on the interlock line of the laser. Since the interlock circuits are connected in series, only one will need to open to open the circuit to the laser.

The interlock system is also designed to accept an 'active' signal from the laser head. When this signal is present, the interlock system will engage the flasher circuitry. Then the warning lamps will flash; they will alternate between half and full power instead of maintaining a constant half-power brightness. This can provide an additional safety check; if the lamps are flashing but the interlock is tripped, then something is wrong.

The interlock panel itself is locked and requires a key to gain access to the internal wiring. In addition, the wiring to the lamps and buttons is enclosed in flexible metal conduit. This is meant to discourage unauthorized personnel from tampering with the interlock wiring. The conduit is shared by the room lighting; however, its power branches off at the point where the power cord enters the interlock panel.

3.2.12. Pulse Generator

The pulse generator is designed to send a pulse signal to the laser upon receiving a triggering RS-232 signal. The pulse signal is meant to be much more controlled than the signal coming from the computer system. When the laser is in pulse mode, it accepts TTL signals through its SMB connector modulated at frequencies up to 150 MHz. The pulse generator outputs such a TTL-compatible signal through its BNC coupler, which is connected to the laser via a BNC-SMB adapter.

The trigger signal coming into the pulse generator should be at least 2 microseconds in duration. This does not pose a problem. However, the triggering signal may be in the high state arbitrarily long; its duration and timing are poorly controlled due to the nature of the control software and the computer system. The circuit was therefore designed to work properly in this situation. It uses an op-amp as a comparator on the input to transform the RS-232 voltages into something more in line with TTL signals, and then passes the output from the op-amp into a simple RC-based high-pass filter. The signal passing through the filter is very short, on the order of 1 to 2 microseconds, and can be used as a triggering signal in an NE555 timing circuit.

The NE555 is a ubiquitous, low-cost precision timer IC. When two NE555 timers are placed on a single IC, the product is the NE556 dual timer. Both the NE555 and NE556 come in a number of variants. The timer used in the pulse generator was a RadioShack LM556 that was on hand at the time of assembly. It has the capability of directly driving TTL circuitry, can source or sink up to 200 mA [31], and can do timing down to microseconds.

Only one of the timers in the LM556 is used in the pulse generator, and it is used in monostable (single-shot) mode. This means that the timer must be triggered separately for each output pulse; it does not run as an oscillator. The duration of the output pulse is determined by the values of a resistor and capacitor connected to the IC. The capacitor used in the pulse generator has a fixed value, while the resistor is a potentiometer connected to a knob. This allows the duration of the pulse to be easily changed and set for the experiment. The output pulse can therefore be set from about 4 to around 120 microseconds by rotating the knob.

The pulse generator also includes connectors meant for use in observing the output pulse. This makes connecting an oscilloscope to the system a simple task. One connector connects through a 1000-ohm resistor to the output pin on the timer, and another connects through a 1000-ohm resistor to the output line downstream of the impedance-matching resistor. The laser head and coupling cables are nominally a 50-ohm impedance system, and therefore the pulse generator should have its output impedance matched to that. However, there are other considerations as well. Tolerable signal voltage and short circuit protection were considered in the design, and so the impedance match is somewhat less than perfect. Nevertheless, the degree of imperfection is not considered to be much of an issue at the operating frequencies in the experiment.

3.2.13. Oscilloscope

The oscilloscope used in the experiment is a Tektronix ® TDS 2014C with Tektronix TPP0201 probes. It nominally responds to frequencies up to 100 MHz, and has four independent channels for input.

The oscilloscope can communicate with a PC using its built-in USB interface. It comes with a limited software suite, but its use is not restricted to the included software. The oscilloscope uses the VISA (Virtual Instrument Software Architecture) software interfaces and a documented command set. This allows custom software to communicate with the oscilloscope. However, such software has not been developed for this experiment; the oscilloscope was used as a stand-alone piece of lab equipment, and the results were manually read and recorded.

3.2.14. High Speed Camera

The high speed camera used to capture the high-speed footage of the experiment is an IDT X-Stream™ VISION XS-4. It has 2 GB of onboard memory and can capture grayscale images. Its resolution is 512 x 512, and at that resolution can capture up to 5,145 frames/second; at certain lower resolutions, it can capture more. It uses a USB interface to the computer system, which runs IDT X-Vision version 1.13.01 for saving the footage and controlling the camera.

The lens used for the camera is a Nikon Micro-Nikkor ® 55mm f/2.8 lens, attached via a Nikon ® F-C Mount Lens Adapter. The entire assembly was mounted to a platform independent of the experiment, and is outside one of the containment box. It is aimed through one of the windows and focused on the bulb in the test section.

3.2.15. Data Acquisition and Control System

The computer used in the experiment is a Dell ® OptiPlex™ 745. It is used for data acquisition and for control of the experiment through the SpeedControl software developed specifically for those tasks. While the system is not meant for high-performance, it does have suitable characteristics for use in the experiment.

It has 2 GB of RAM and, more importantly, a 2.4 GHz Intel ® Core 2 Duo™ CPU E6600. The multi-core characteristic is important; the SpeedControl software is multithreaded and one of the threads uses a full CPU core as long as the program is running. This is due to its practice of polling the serial port for high-speed observation and control; the main thread cannot afford to sleep (give up large parts of its timeslice while the CPU does other things). The presence of a second CPU core gives the other threads and other background tasks system resources in which to execute, largely without interfering with the time-critical thread.

The system runs Microsoft ® Windows™ XP with Service Pack 3 and is regularly updated. Since Windows™ XP is a preemptive multitasking system, some care needed to be taken in the development of the control and data acquisition software to ensure that time-critical operations do in fact occur in a timely manner, and that the preemptions that do occur are properly taken into account.

3.2.16. SpeedControl Software

The SpeedControl software provides an integrated package for performing the functions of instrumentation, control, and data acquisition for the experiment. It performs low-level control for the experiment and automates a number of higher-level

functions. It includes a User's Guide; this document can be accessed any time the SpeedControl software is running by clicking on the 'Help' button.

A significant development effort went into making the SpeedControl software functional. It was written in Microsoft ® Visual Basic™ 2005 Express as a .Net™ 2.0 application. It does make direct use of certain Win32 API calls, so its cross-platform compatibility is somewhat limited. It requires a computer running Microsoft ® Windows™ 2000 or higher with .Net 2.0™ installed. It also requires more than one CPU core in order to run properly, and benefits greatly from faster CPU performance. The presence of at least one serial port is required as well.

3.2.16.1. Using the SpeedControl Software

Upon launch, the User is presented with a loader screen. This lets the User select the port(s) for use with the speed controller and the laser system if present; it can be the same port as the speed controller. It is also where the User selects the output logfile, giving the option of appending or replacing any existing file of the same name and location. In addition, the choice for the type of speed measurement calculations is made in the loader screen.

Once the selections are made, the program launches the necessary threads in its startup process. Once all the appropriate threads have been launched and the initialization sequence completes, the software will enter its running mode. The User is presented with a split window that gives options for speed control and laser firing sequences, with a glimpse at the logfile output and laser I/O; this window is presented in

Figure 18. In this mode, the main execution thread will consume as many CPU cycles as it can on one CPU core.

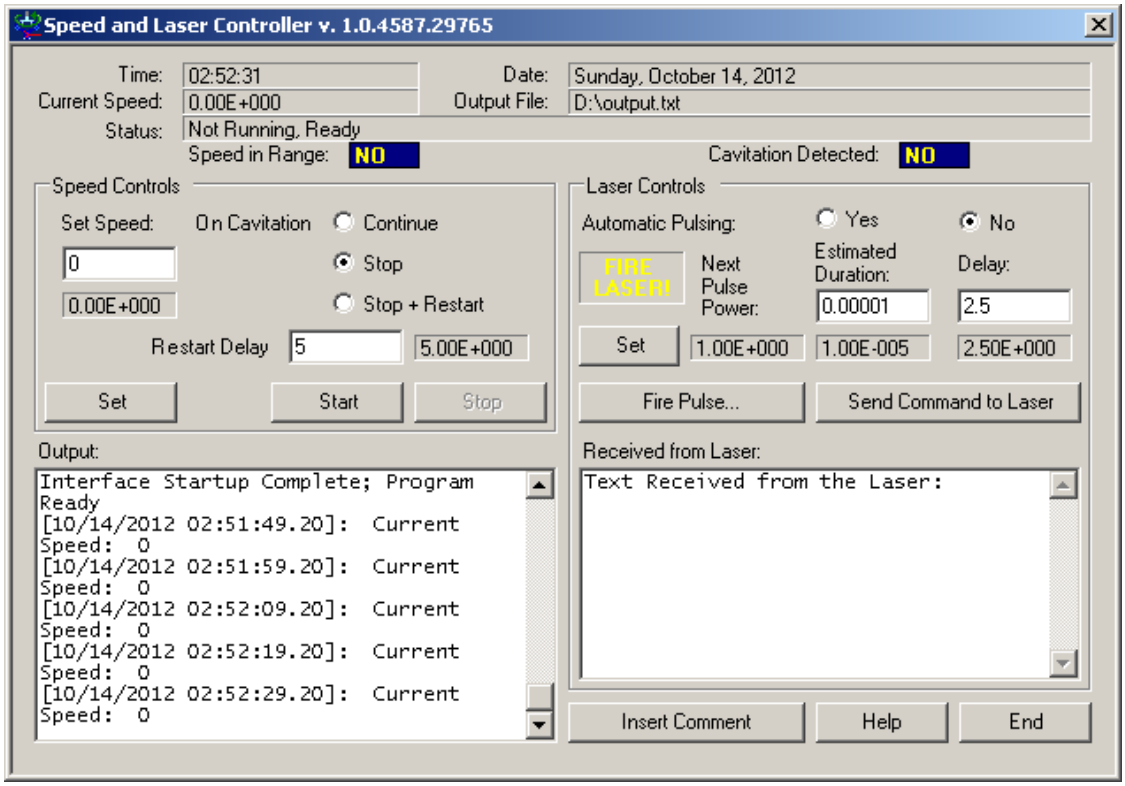

Figure 18: Main SpeedControl Window

In the main operating mode, the software will monitor the experiment's speed, cavitation status, and will record data received through the serial port used for the experiment; if the laser is set to jointly use the port, it will be acknowledged upon reception of transmitted data. The software will also work to maintain the set speed, signal laser pulses as appropriate, and transmit the appropriate data to the laser head. In addition, the software logs the relevant data and maintains a user interface with current information.

From the User Interface window, the User can easily monitor the status of the experiment. In addition, simple controls are provided for the user-adjustable options available at runtime. Speed control is accomplished on the left side of the window; the User simply types in a value for the speed in rotations per second in the appropriate box and clicks on 'Set' beneath it to set the value. The User can also change the program's actions when cavitation is detected (continuing, stopping, or pausing and resuming) and a pause duration if the cavitation action is set to pause and resume the rotation of the test section. The User can also click 'Start' to start running the experiment; it then gets grayed out and the 'Stop' button is activated. Pressing 'Stop' halts the experiment and terminates laser pulsing.

If the software is running with the laser control enabled, such controls can be accessed on the right hand side of the window. An estimated duration of the pulse can be entered directly, as can the time delay between pulses. These new values are not set until the User clicks on the 'Set' button in the laser control area. Additional control and options can be accessed via their relevant buttons. Text commands can be sent directly to the laser head through the window brought up when the User clicks on the 'Send Command to Laser' button. A manual pulse can be issued through the window brought up when the User clicks on the 'Fire Pulse...' button. It also allows the User to change the starting power for automatic pulsing by setting the power for the manual pulse and closing the manual pulse window.

The User can also insert a comment into the logfile by clicking on the 'Insert Comment' button in the main window and typing in the comment. The nearby 'Help' button brings up the User's Guide, and the 'End' button will exit the program.

3.2.16.2. SpeedControl Software Architecture

The software manages to do much of its operation in its main thread. In the main operating mode, it loops endlessly without sleeping. This is so that it can get timing issues handled as well as it possibly can. It does, however, allow certain cooperative OS functions time at the end of each loop executed. It uses an elevated priority state to help ensure that timing is not thrown too far off by preemptive OS functions. Timing is critical in the main thread because it directly handles the logic states of the input sensors and directly controls the output states, which can have a desirable update or monitoring frequency on the order of thousands of times each second; the timeslices in Windows do not have sufficient granularity to properly handle sub-millisecond sleep intervals.

The main loop of the main thread does a number of things in each loop. This includes

- Clock functions for precision timing and the detection of significant 'dead'/preempted times
- Reading the input state for speed and cavitation sensor, and processing the information
- Setting the speed control pin as appropriate

- Processing the Laser Fire Control Sequence, which can override speed control values, issues commands to the laser head, and controls the status of the fire control pin on the serial port
- Determines if enough time has elapsed for the less-frequent tasks to be executed
- Allowing Windows to process certain events without fully releasing the timeslice

The less-frequent tasks include

- Speed logging
- Reading and Writing data to and from the serial port data buffers
- Updating the User Interface
- Collecting User-Controlled values
- Additional speed control calculations
    - Predicting the speed in the near future with simple linear regression
    - Determining if the speed is within tolerance limits
    - Determining a time fraction for the speed control pin to be in the 'high' state
- Processing certain delays

The speed measurement is based around a simple tachometer concept: counting how frequently the light and dark regions on the main shaft go past the speed sensor.

Every time the software determines that the sensor changed state (light to dark or dark to light), the appropriate counter is incremented. In addition, it uses a sort of 'running average' scheme using exponential decay to keep the transition count and live time up to date. If a significant 'dead' period is detected (a missing chunk of time long enough to mask two or more transitions at the current speed), that time period is removed from consideration for the speed calculation if the 'Advanced Timing Feature' is enabled. The speed determined in the calculation is logged at a minimum frequency, when certain events take place (laser firing, entering or leaving the tolerance ranges), and when the speed has had a significant change from its last recorded value.

Determination of the cavitation status is simpler than the speed calculation. It looks for a signal from the cavitation sensor of a minimum duration (to eliminate spurious signals) to determine whether or not a vapor column has formed in the bulb of the test section. If the minimum duration requirement is met, the cavitation state is considered to be 'true' and the event is logged.

Management of the speed control output state grew from a relatively simple and straightforward method to a more complex set of calculations. On its face, the concept is still simple: the pin should be in a 'high' state for a determined fraction of the time, and in the 'low' state for the remainder of the time. This is meant to occur on very short timescales so that this signal gets converted to an analog voltage on a capacitor in the controller electronics with little ripple; it is much like Pulse Width Modulation at variable frequencies. This control fraction is set with a simple concept: raise the value when the speed is low, and reduce it when the speed is high. However, the method of

achieving this simple concept became somewhat complex before a working solution was developed.

The solution required a degree of responsiveness to random fluctuations and at the same time have the speed remain stable. In practice, making the system more responsive produced odd behavior; oscillations were seen, and the speed could swing rapidly and uncontrollably from zero to full speed. Reining in the oscillations reduced the responsiveness of the system, and random fluctuations in the speed would end up dominating; the speed would never completely approach the desired value.

To address those issues, the calculations for determining the speed control fraction take several variables into account. One is a prediction of what the speed will be in the near future. Another is a running average of the control level as an analog to the voltage across the control capacitor in the speed controller electronics. In addition, 'dead' times are taken into account. Such periods can result in significant deviations in the output; the actual output fraction can differ greatly from the desired fraction. These deviations are detected and incorporated into the value of the fraction for the next time period. However, a simple calculation does remain at the core of the determination of the output fraction: the change in the fraction is equal to the sum of the speed deviation multiplied by a coefficient and the cube of the speed deviation multiplied by another coefficient. The coefficients are such that the cubic term comes to dominate when the difference between desired and measured speed is very high; otherwise, the linear term dominates.

The values for the speed control fraction are used for a fraction of a second before they are updated. Within their use period, the speed control output is high if the fraction to that point is lower than desired; it gets set low if the time fraction is greater than desired. Precision timing is therefore critical for proper control of the speed control signal.

The Laser Fire Control sequence addresses and automates key parts of the experimental procedure. It takes the procedure to adjust and fire each pulse and breaks that procedure down into a series of 'stages' where each stage takes certain actions and will only proceed to the next stage once certain criteria are met. It can take a large number of cycles through each execution loop for the criteria for moving on to the next stage to be fulfilled; a stage can take an arbitrarily long time to complete. Therefore, the software was designed so that the code in each stage would not halt the loop's execution. For example, a stage that consists of a delay for a certain amount of time will simply compare the system clock to a timer value each time the main loop is executed. If insufficient time has passed since the timer was started, the software will return to executing the main loop without advancing to the next stage. When the timer value has expired, the software will execute the following stage the next time through the main loop.

The concept for the firing sequence is simple: halt the rotation of the test section, adjust the laser power, bring the test section back up to speed, issue a laser pulse, and finalize the sequence. There are a series of delays in that sequence to ensure that everything happens in the correct order. If, for example, the laser processed its power

setting command slowly and power was changed after the test section started rotating again, it might cause an inadvertent cavitation event. This is because the laser emits radiation while it is adjusting its power in pulse mode.

The sequence for a 'manual pulse' is essentially as described above. An 'automatic pulse' is similar, but does have several notable differences. An automatic pulse sequence, when cavitation is detected, will end. In addition, after the laser power is adjusted and it returns to speed, more than one shot will be fired. The software will pulse the laser three times at that particular power with a delay between shots. Afterward, there is a delay to see if a vapor column develops. Then, if no vapor column develops, the laser power is incremented by 1 mW and the sequence starts over. This repeats until the firing sequence is interrupted by cavitation, a timeout, user intervention, or the next pulse power exceeds the laser's capabilities. In this manner, the software automates the experiment. It ramps up the laser pulse power until it finds the power necessary to induce cavitation in a test section rotating at its set speed with its particular fluid fill parameters.

The threads that the SpeedControl software executes, other than the main thread, have somewhat mundane tasks. One thread takes the information to log from a buffer and writes that data to a logfile. Another one acts as an intermediary between the main thread and the User; it controls the user interface and communicates information back and forth between the main thread and the UI window. An additional thread only runs during startup with laser control activated; it sends the initialization sequence to the laser head. In addition, if the laser port is active but set to be different from the main serial

port, its I/O will be handed off to another thread. There is also an underlying monitor thread that only plays a role during program startup and shutdown, and when threads terminate unexpectedly. It launches the relevant threads at startup, periodically checks to make sure they are running during the main operation loop, and attempts to close resources gracefully on shutdown or when things go awry.

3.3. FLUID CHOICE AND SEEDING

The liquid used in this experiment was hardware-store grade acetone. While it is understood that such a grade of acetone may be full of impurities, that is not expected to be one of the primary sources of error in the experiment. The acetone is seeded, and the seeding material is expected to override any impurities that may be present in such low-grade acetone.

The original intent was to use a pure liquid without any seeding material. This would encourage the use of Eq. (62) to determine localized energy deposition in the liquid. However, early testing suggested that the laser system was not powerful enough to deposit sufficient localized energy in the tensioned acetone to guarantee an induced cavitation event and the subsequent formation of a vapor column. Therefore, a material was needed that could, when mixed with the acetone, provide weak points within the liquid in order to seed the nucleation of bubbles and enhance the local energy absorption of the liquid. Black toner from a used toner cartridge was found, through trial and error, to have some suitable characteristics.

The toner had small particles that would readily disperse into the acetone, and a significant fraction appeared to stay in suspension for long periods of time. It also

absorbed significant fractions of the incident visible light; as 403 nm is nominally within the visible spectrum, it was thought to provide excellent local absorption and could then transfer the absorbed radiant energy to its surrounding fluid. Since the toner appeared to disperse in the acetone as a suspension rather than by dissolving, the energy absorption should be highly localized to the suspended particles rather than evenly dispersed in the acetone.

Such a combination was thought to be helpful; the particles would provide local weaknesses in the liquid, as the adhesive forces were expected to be less than cohesive forces in the acetone. In addition, the laser energy would be preferentially available to those weak areas, as the absorption by the toner particles (appearing black to the eye) is much greater than that of acetone, which is transparent. As a result, it was expected that this would decrease the demand for laser power in the experiment. However, it may provide a major source of error, and discourages the use of Eq. (62). As a result, it was assumed that all (or at least a constant fraction of) the absorbed laser energy would be made available at the interface between the toner particles and bulk acetone, encouraging the direct use of the fluence equations with a simple scaling coefficient.

3.4. EXPERIMENTAL PROCEDURES

The performance of an experiment can be broken roughly into three distinct phases: Pre-Operation, Operation, and Shutdown.

3.4.1. Pre-Operation Procedures

The Pre-Operation procedures are largely independent of data runs. They involve certain systems checks and preparations.

1. Prepare bulk test solutions for future use
    a. Measure the mass of each component
    b. Mix and store them in CLEAN, AIRTIGHT containers
    c. Ensure the containers are properly labeled
2. Check the functionality of the laser interlock system according to the Standard Operating Procedures for the laser system
3. Power on the RS-232 isolator and check that the speed sensor calibration is good. If not, adjust the calibration knob
4. Don the protective laser goggles
5. Prepare the room for the laser interlock to be armed
6. Open the shutter on the laser
7. Power on the laser
8. Open the control software provided by the laser vendor
9. Adjust the laser to low-power (less than 5 mW) CW; do not turn on the beam
10. Arm the interlock system and enable the beam
11. Check the rough alignment of the beam with an index card or something functionally similar. It should be directed toward the location of the bulb in the bottom of the test sections; if it is far off, perform a rough alignment.
12. Disable beam emission and shut down the laser.

3.4.2. Data Operation Procedures

These procedures come immediately after the Pre-Operation Procedures, and may be repeated several times for a day of data collection.

1. Select the test section for use and clean it. This may be difficult to accomplish.
2. Clean the syringe that will be used to handle the fluid.
3. Choose a fluid solution prepared previously in the Pre-Operation Procedures. If necessary, agitate the solution to ensure dispersal of anything that may have settled out.
4. Withdraw a sufficient amount of solution to fill the clean test section using the clean syringe
5. Fill the test section incompletely; the fluid should be in the upper arms but filled no further than the outer edge of the PVC mount.
6. Mount the filled test section onto the experiment's main rotational shaft; record the bulb and fill fluid used.
7. Making sure that the protective laser goggles are worn, power on the laser.
8. Using the vendor's control software, set the laser to low power (< 5 mW) and check the fine alignment. This is easiest to accomplish if the test fluid fluoresces; the hardware-store grade acetone does
    a. The focus of the beam should be near the center of the bulb, along the axis of rotation, which may not match the bulb centerline.

b. Close the box, power on and arm the box interlock, power on the speed controller and set the switches to manual control.
c. Adjust the rotation rate to a low speed, but high enough to visually determine the axis of rotation.
d. Ensure that the focus does correspond to a point along the axis of rotation sufficiently far above the bottom of the bulb that cavitation events will not be generated there. Adjust if necessary
e. The alignment should work for several tests using the same test section; it should not be tweaked needlessly

9. Adjust the calibration of the cavitation sensor
    a. Making sure that there is no vapor column in the bulb, find the knob position for the threshold between "detected" and "not detected." The test section should be rotating at a low rate for this. Note the position of the knob.
    b. Ramp up the speed and induce cavitation (this may require use of the laser; the vendor's software can be of assistance).
    c. Reduce the speed until the vapor column in the bulb is very small.
    d. Adjust the calibration knob on the cavitation sensor to find the threshold position for detection of a vapor column; note the position of the knob.
    e. Set the position of the calibration knob on the speed sensor to halfway between the two thresholds just determined

f. NOTE: since the sensor uses a photocell and incident light passing through the test section to determine its state, extraneous or changing illumination may interfere with its correct operation. It may also have trouble if the liquid absorbs significant amounts of light in the red frequencies.

g. Set the rotation rate back to zero.

10. Degas the fluid in the test section

a. Check to see if the fluid can maintain the maximum rotation rate deliverable without forming a vapor column by ramping up the speed to its limit.

b. If a vapor column forms, stop the rotation, allow the gas bubble to escape, and repeat the ramp test until a vapor column no longer appears at the maximum rotation rate.

c. Ramp the speed up to its maximum value.

d. Briefly set the laser power to 100 % in CW mode using the vendor's software. Once a vapor column forms, set the power to zero

e. Slow down the speed until the vapor column is little more than a bubble in the bulb

f. Maintain this condition for a few minutes to attempt to diffuse any gas remaining in the liquid out to the vapor column, which should be largely the vapor of the liquid rather than dissolved gas.

g. Repeat as necessary

h. Stop the rotation

11. Exit the laser vendor's control software

12. Determine a pulse duration to be used for the subsequent tests

    a. Disable laser emission.

    b. With the oscilloscope attached to the pulse generator's output, feed a square wave into the input of the pulse generator that is compatible with RS-232 voltages and signaling.

    c. Adjust the timer knob on the pulse generator until the pulses it outputs match the desired pulse duration.

    d. Reconnect the pulse generator to the appropriate serial port line

13. Make sure the laser is powered on

14. Start the SpeedControl software, and set its initial parameters

15. Once finished starting up, make sure that the software is set to not control the speed.

16. Switch the electronic speed controller over to external input for speed control.

17. Perform a test sequence.

    a. Using a micrometer, measure the distance between the menisci in the arms of the test section. Record the value.

    b. Close the box and arm its interlock.

c. Set the initial laser pulse power and duration in the SpeedControl software, and enable automatic pulsing. The initial laser pulse power need not be zero if it is known that cavitation will not occur below certain values.
d. Set the desired rotation rate in the SpeedControl software, and set the control to stop once cavitation is detected.
e. Start the control, and allow it to automatically perform its test sequences.
    i. First, the program will stop the rotation and calibrate the laser pulse power.
    ii. Then, the test section is brought back up to its desired speed.
    iii. Up to three pulses are issued to the laser
    iv. If no vapor column forms, the laser power is incremented, the speed brought to zero, and this sequence is repeated
    v. If a vapor column forms, then the speed is brought to zero and the sequence ends.
f. Upon a successful cavitation event, determine the measured duration of the pulse issued to the laser from the pulse generator. Record this value.

g. Check the logfile and determine the rotation rate at the time the critical pulse was issued; record this value as well as the laser power for the pulse.

18. Perform additional test sequences as necessary and desired.
19. Once a test section has been filled with the testing fluid, it should not be topped off; it should be completely drained and rinsed before refilling.

3.4.3. Shutdown Procedures

These procedures are primarily for the end-of-the-testing-day cleanup.

1. Set the speed controller to manual and adjust it to zero.
2. Power off all the systems that were used, except the laser interlock.
3. Once the laser has been powered off, close the shutter on the laser head.
4. Power off the laser interlock. The laser goggles can now be removed.
5. Dispose of the used testing fluids as necessary.
6. Backup the saved test data.

3.4.4. High-Speed Camera Notes

When a high-speed camera run is performed, it is much like a normal test. However, some additional "juggling" is involved. Before running the test, it should be aimed and configured in its software for the appropriate framerate, resolution, etc. It may be necessary to provide additional illumination; care should be taken to ensure this does not interfere with the cavitation sensor.

Since the camera has limited onboard memory, it cannot be run for the entirety of the test sequence. Instead, the user must carefully watch the indicators on the

SpeedControl software in the background while in the X-Vision software. Shortly before a laser pulse is issued, the user must trigger high speed acquisition from the X-Vision software. Depending on the settings, it may be necessary to do this for each pulse. Once a cavitation event has been detected, the user can save the high speed acquisition to the DAQ computer in a variety of formats.

Due to the lack of high-speed camera support in the SpeedControl software, it is recommended that it only be used in special runs. The recommendation is that instead of doing a ramp in laser energy to find the cavitation threshold, it be deliberately set WELL ABOVE the cavitation threshold to ensure a video capture of the event and the formation of the vapor column.

3.5. EXPERIMENTAL RESULTS

Several tests were successfully performed with the existing experimental setup. One successful high speed camera run was recorded, and four separate fluid fills were done for threshold data runs. All runs used the same test section, recorded as Bulb A. The four threshold data runs drew from the same mixture of acetone and toner, recorded as Mixture A. However, due to particulate settling, they may not have had identical fills. In addition, the high speed camera run used an uncharacterized toner/acetone mixture.

The common acetone/toner mixture is relatively clear. The dispersed particles do not do much to darken the mixture, except when they settle out. The original mixture used 145.0028 g of acetone, into which was mixed 0.10919 g of collected toner particles. This amounts to a ratio of 0.753 mg toner per g acetone. However, the larger and heavier toner particles do not remain suspended long, while the very smallest ones

should remain in suspension indefinitely. Indeed, it was seen that the largest of particles never fully dispersed into the liquid, and instead quickly settled onto the bottom of the container. Over the course of hours to days after agitating the mixture and subsequently allowing it to stand undisturbed, additional particles were seen to have settled out on the bottom of the container. However, the liquid never regained its complete original clarity, indicating that some material had not come out of suspension/solution.

3.5.1. High Speed Visualization

This test was intended solely to provide visualization of a cavitation event and the early evolution of a vapor column. As such, the particulate concentration was not determined, and neither was the operating centerline tension; it is estimated as having been around -2 atm ± 1 atm based on an apparent mid-level liquid fill. It operated at 200 rps, and the recording framerate was 1000 fps.

The state before and during the laser pulse can be seen in Figure 19 and Figure 20, respectively. A small white dot appears, and a vapor column grows out from there. The subsequent formation and stabilization of the vapor column is shown proceeding in time in Figure 21, Figure 22, Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28, Figure 29, Figure 30, Figure 31, Figure 32, and Figure 33. They show the state of the bulb at 1, 2, 3, 4, 5, 6, 7, 9, 11, 16, 21, 41, and 191 ms after the laser pulse, respectively.

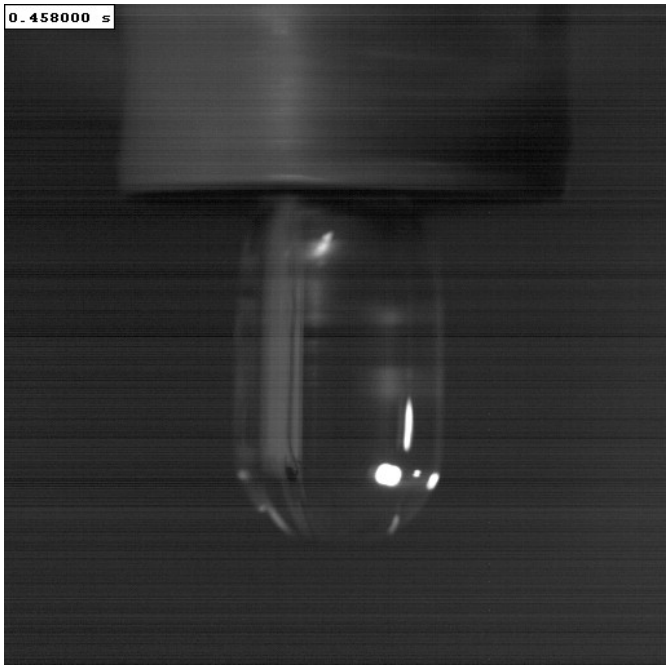

Figure 19: Before the Pulse

One thing that should be recognized is the presence of reflections from the light sources for the camera. They can be seen in all the high-speed captures, and should not be confused with the phenomena of interest, especially in Figure 20. The bright reflections in Figure 20 are very much like those in Figure 19, occurring primarily in the bottom right and top left of the bulb; less prominent reflections occur elsewhere. The spot in Figure 20 slightly to the left of the bottom center of the bulb, however, is not a reflection and is unlikely to be a camera artifact; it is in the same location as the growing disturbance in Figure 21.

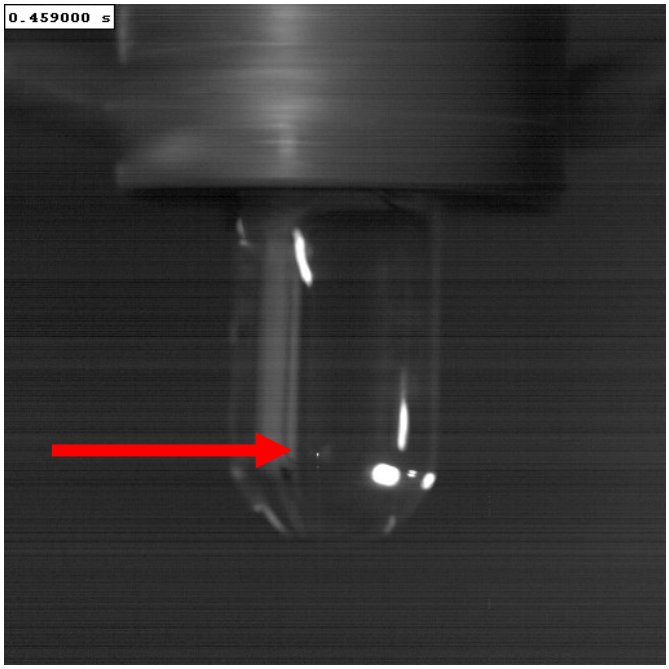

Figure 20: Firing the Pulse

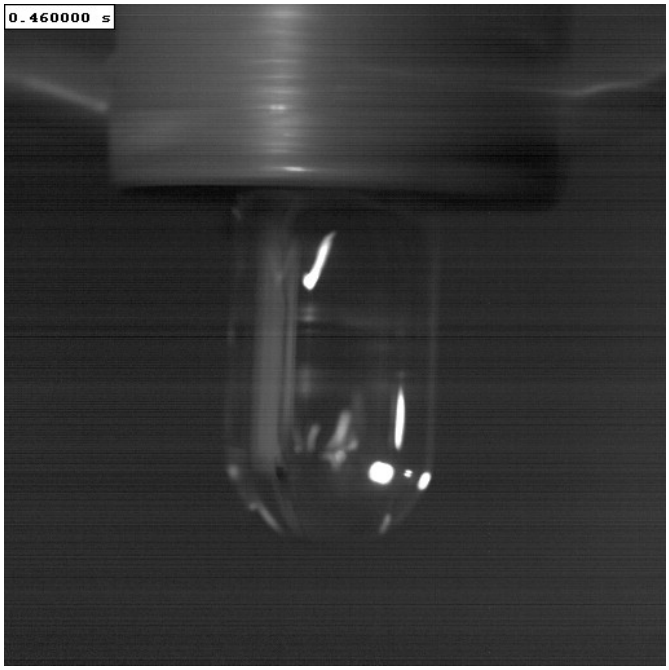

Figure 21: 1 ms Post-Pulse

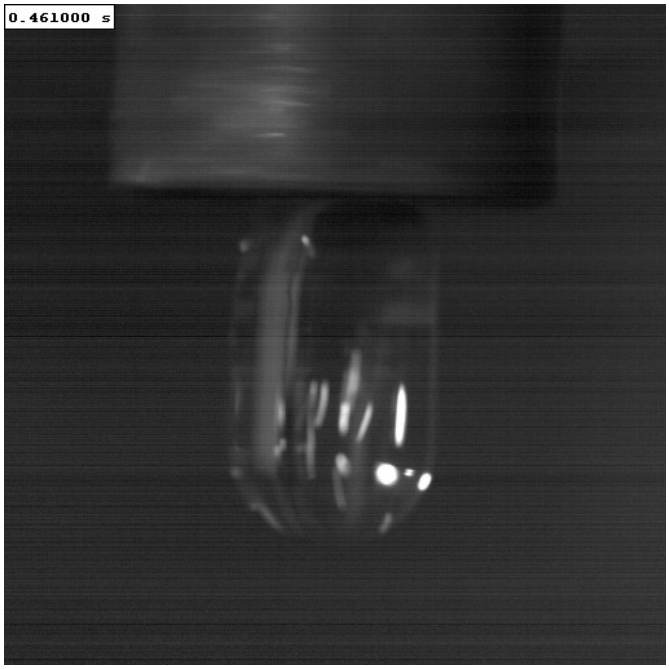

Figure 22: 2 ms Post-Pulse

In Figure 21 and Figure 22, it can be seen that there is a rapidly expanding region in the bottom of the bulb growing out from the initial spot. This region begins to take shape in Figure 23 and Figure 24 as a column in the bulb. However, it is still in its initial growth phase; momentum expands it past its long-term (stable) boundaries. The approximate stable boundaries can be seen in the image taken 191 ms after the laser pulse.

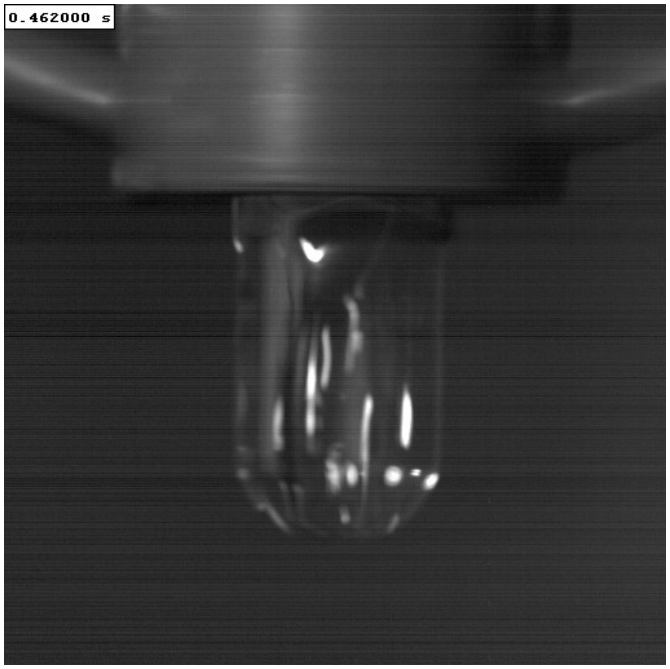

Figure 23: 3 ms Post-Pulse

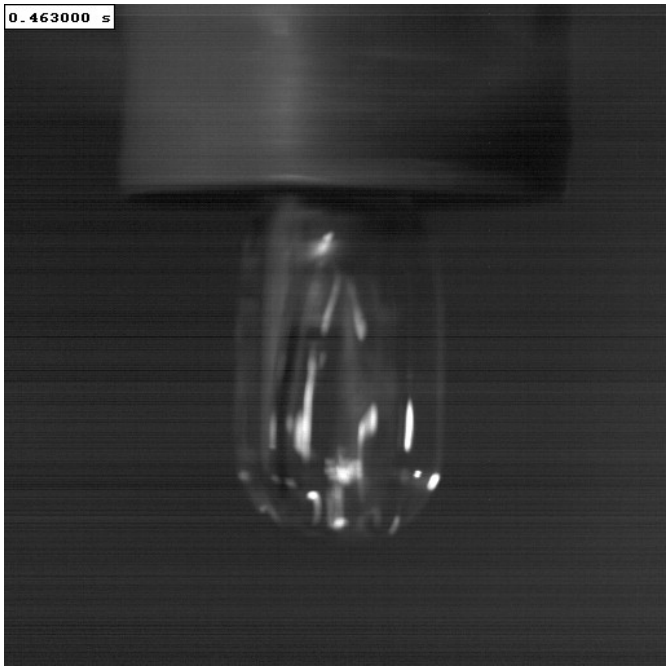

Figure 24: 4 ms Post-Pulse

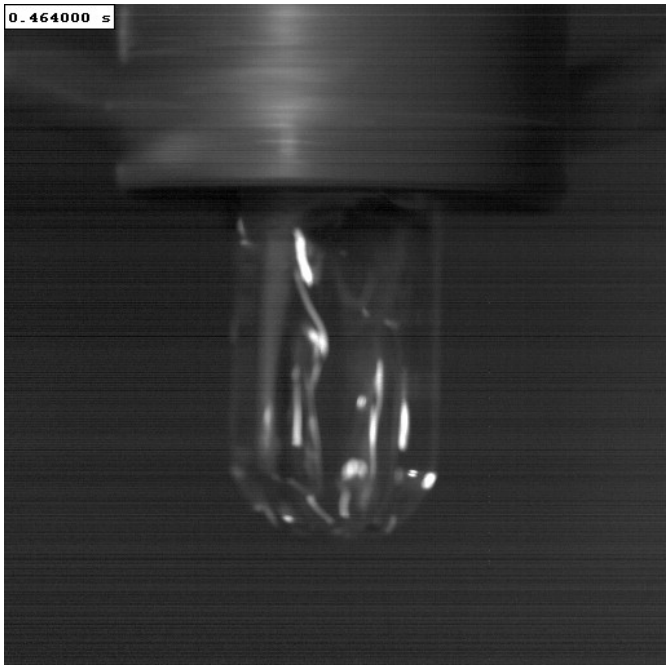

Figure 25: 5 ms Post-Pulse

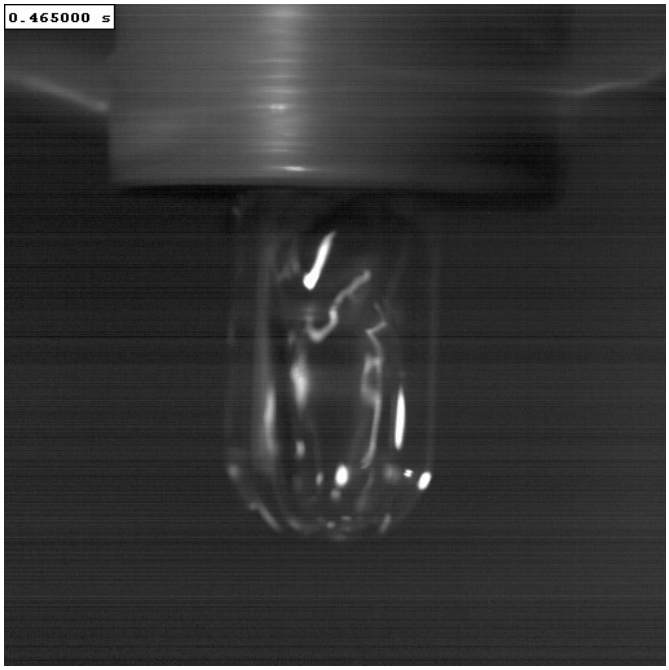

Figure 26: 6 ms Post-Pulse

In Figure 25 and Figure 26, the initial growth phase of the vapor column is winding down, and it shifts back towards its eventual steady-state conditions. However, as seen in Figure 27 and Figure 28, it has a chaotic period of acceleration to undergo. The liquid at the edge of the new vapor column was originally in the vicinity of the centerline, and had a low tangential velocity. When the liquid moves out to a new radius, its tangential velocity is essentially unchanged, and it needs to accelerate to the appropriate in order to match the angular velocity of the system. This acceleration of liquid in the bulb results in some chaotic disturbances.

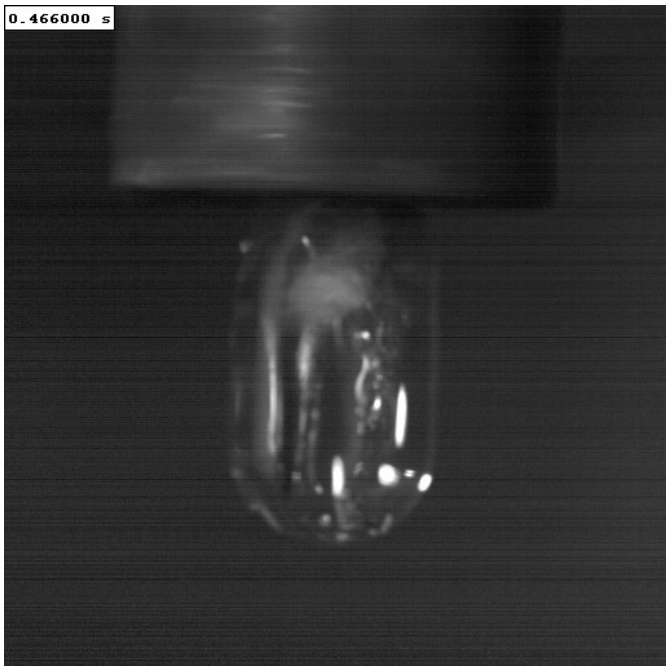

Figure 27: 7 ms Post-Pulse

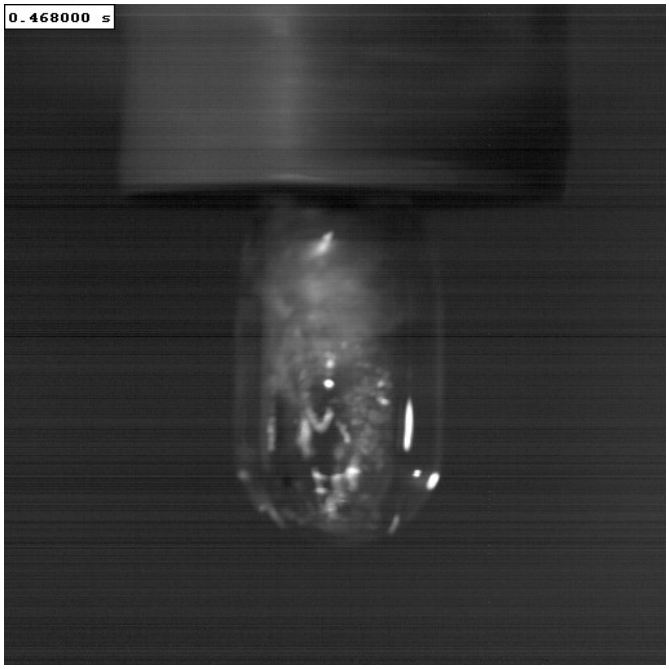

Figure 28: 9 ms Post-Pulse

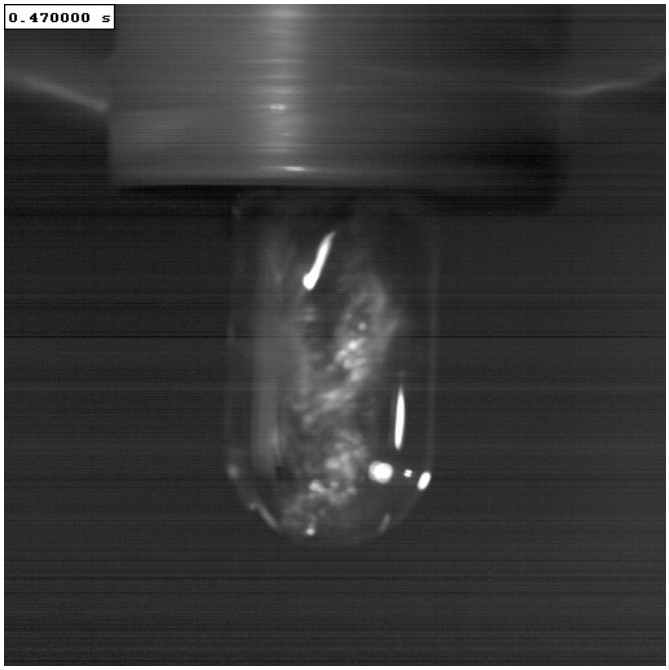

Figure 29: 11 ms Post-Pulse

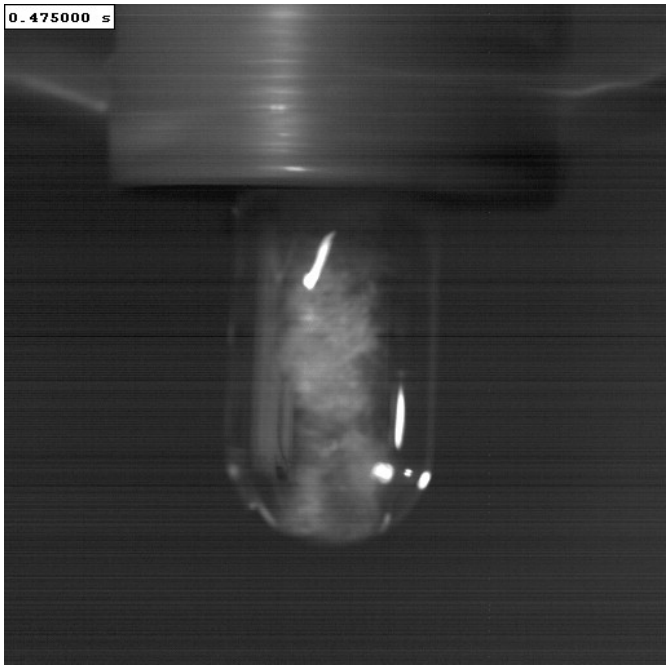

Figure 30: 16 ms Post-Pulse

Figure 29 and Figure 30 show the largely formed vapor column, but the chaos of acceleration is still present. After some time, as seen in Figure 31 and Figure 32, the chaos dies down. Within 16 ms, the vapor column is essentially in its steady state. Vibrations in the equipment play a major role in the remaining chaotic behavior.

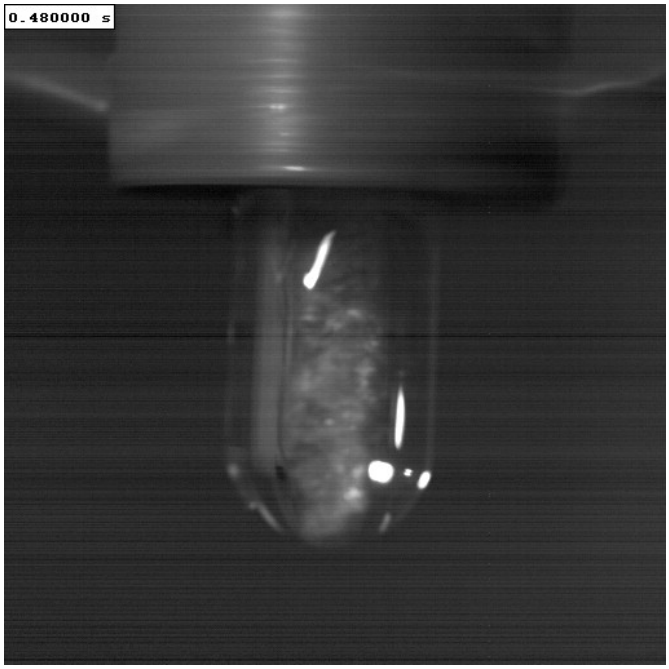

Figure 31: 21 ms Post-Pulse

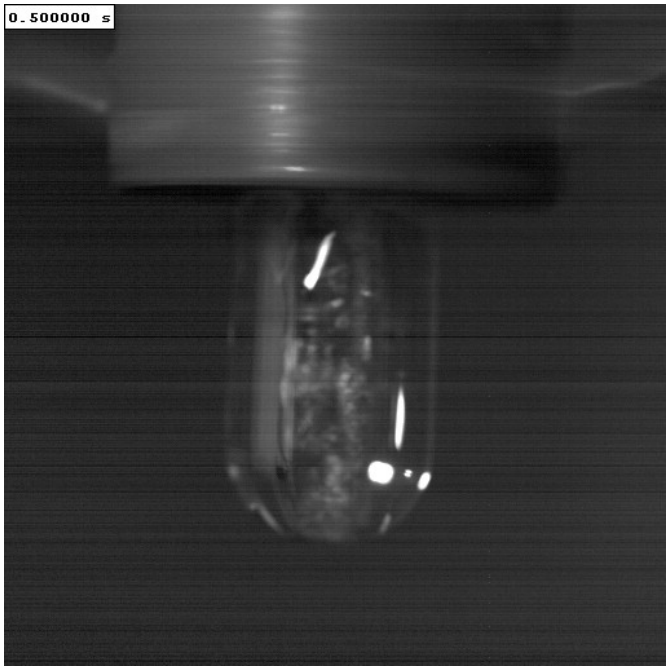

Figure 32: 41 ms Post-Pulse

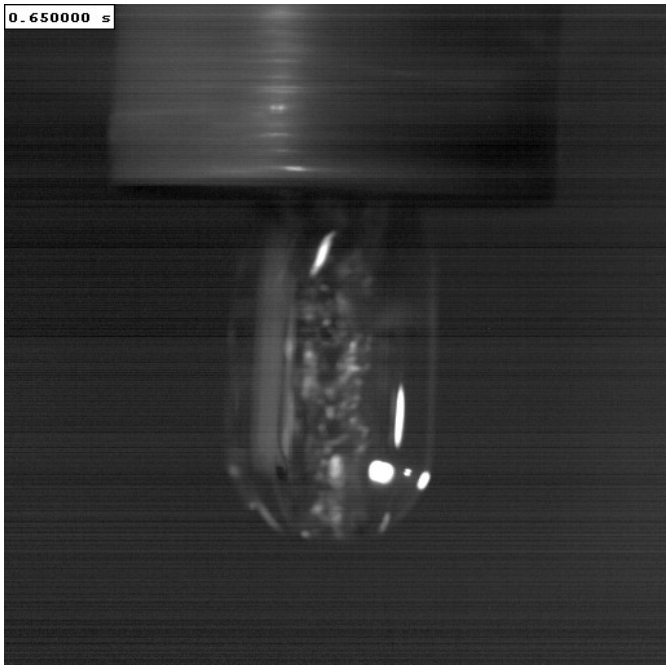

Figure 33: 191 ms Post-Pulse

As seen in Figure 32 and Figure 33, the vapor column after 191 ms is not much different from the vapor column 41 ms after the laser pulse. The steady-state situation can continue with few changes for large periods of time. Long-term, the primary causes of deviations from the steady-state vapor column would include fluctuations in the rotation rate, vibrations/deformations, and the slow disappearance of acetone out the top of the test section as it evaporates out to the atmosphere at large. This occurs on time scales much larger than those used for testing.

3.5.2. First Fill

The first fill was used for a series of 50-μs pulses at 180 rps. In all, 28 threshold tests were successfully completed before the fill was depleted to a level at which it could

not ramp up to speed without the formation of a vapor column. The results are given in Table 3.

Table 3: Data from the First Pulse Series

Test Number	Fill Diameter (mm)	Power (mW)	Pulse Duration (μs)	Speed (rps)	Centerline Pressure (Pa)	Pulse Energy (J)
1	38.83	42	50.4	178.196	-8.40E+04	2.12E-06
2	39.24	42	50.4	180.5	-9.29E+04	2.12E-06
3	40.07	56	50.4	180.88	-1.02E+05	2.82E-06
4	40.45	47	50.4	181.04	-1.06E+05	2.37E-06
5	41.22	51	50	180.12	-1.12E+05	2.55E-06
6	41.73	43	50	180.83	-1.19E+05	2.15E-06
7	42.72	55	50	182.97	-1.35E+05	2.75E-06
8	44.17	43	50	179.91	-1.43E+05	2.15E-06
9	45.1	50	50	180.38	-1.55E+05	2.50E-06
10	45.72	47	50	180.89	-1.63E+05	2.35E-06
11	46.25	52	50	180.89	-1.70E+05	2.60E-06
12	46.8	46	50	182.31	-1.80E+05	2.30E-06
13	46.98	56	50	181.59	-1.80E+05	2.80E-06
14	47.2	50	50	180.88	-1.81E+05	2.50E-06
15	47.95	55	50	179.96	-1.87E+05	2.75E-06
16	48.61	57	50	178.77	-1.91E+05	2.85E-06
17	49.43	51	50	181.28	-2.10E+05	2.55E-06
18	50.21	47	50	181.48	-2.20E+05	2.35E-06
19	50.85	46	50	182.68	-2.33E+05	2.30E-06
20	51.5	41	50	180.69	-2.34E+05	2.05E-06
21	51.8	56	50	181.71	-2.42E+05	2.80E-06
22	52.52	54	50	179.99	-2.45E+05	2.70E-06
23	53.6	51	50	181.07	-2.63E+05	2.55E-06
24	54.09	49	50	181.51	-2.72E+05	2.45E-06
25	55.03	53	50	179.96	-2.78E+05	2.65E-06
26	56.4	56	50	181.21	-3.03E+05	2.80E-06
27	58.76	54	50	181.51	-3.39E+05	2.70E-06
28	63.86	46	50	182.21	-4.23E+05	2.30E-06

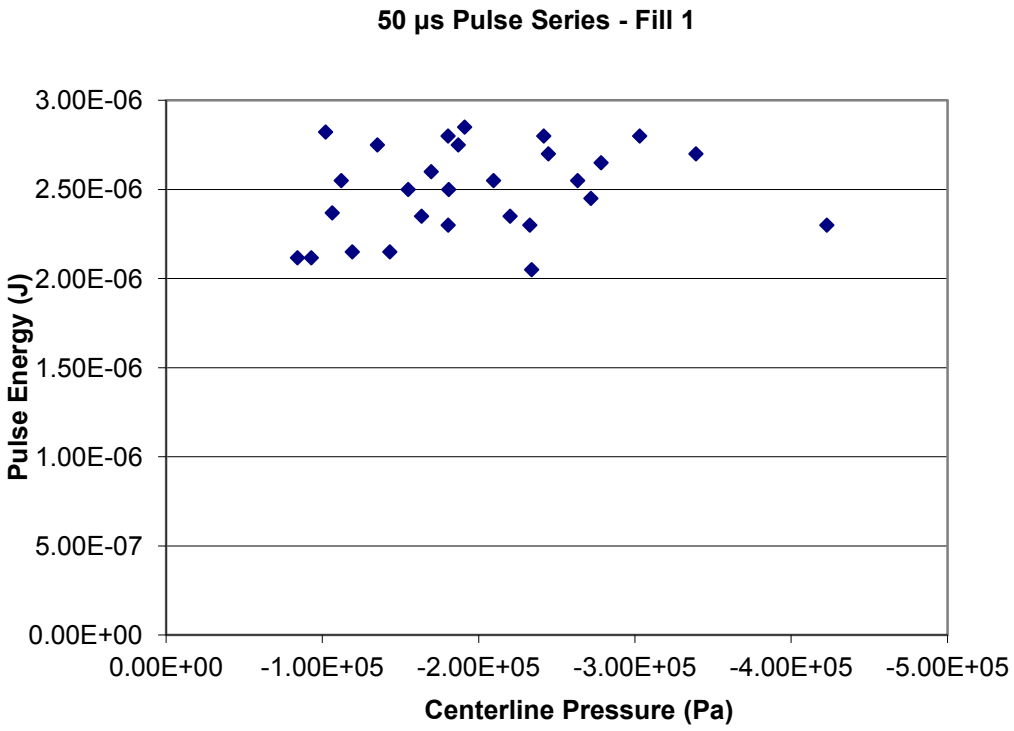

Figure 34: First Fill

The information in Table 3 is represented graphically in Figure 34. As one can see, there is plenty of scatter in the data. In addition, whatever trend there may be is hidden by that noise. It would appear that the trend is essentially flat. This can be the result of thermal diffusion effects. If the pulse is not short enough, significant amounts of the deposited energy can diffuse away instead of contributing to bubble nucleation; the diffusing energy may be so large as to completely mask any underlying trend in the required energy threshold.

3.5.3. Second Fill

The second fill was used for a series of 6-µs pulses at 210 rps. In all, 20 threshold tests were completed before the fill was depleted to a level at which it could

not ramp up to speed without the formation of a vapor column. However, one test was of unusable quality, and an additional four are of questionable quality.

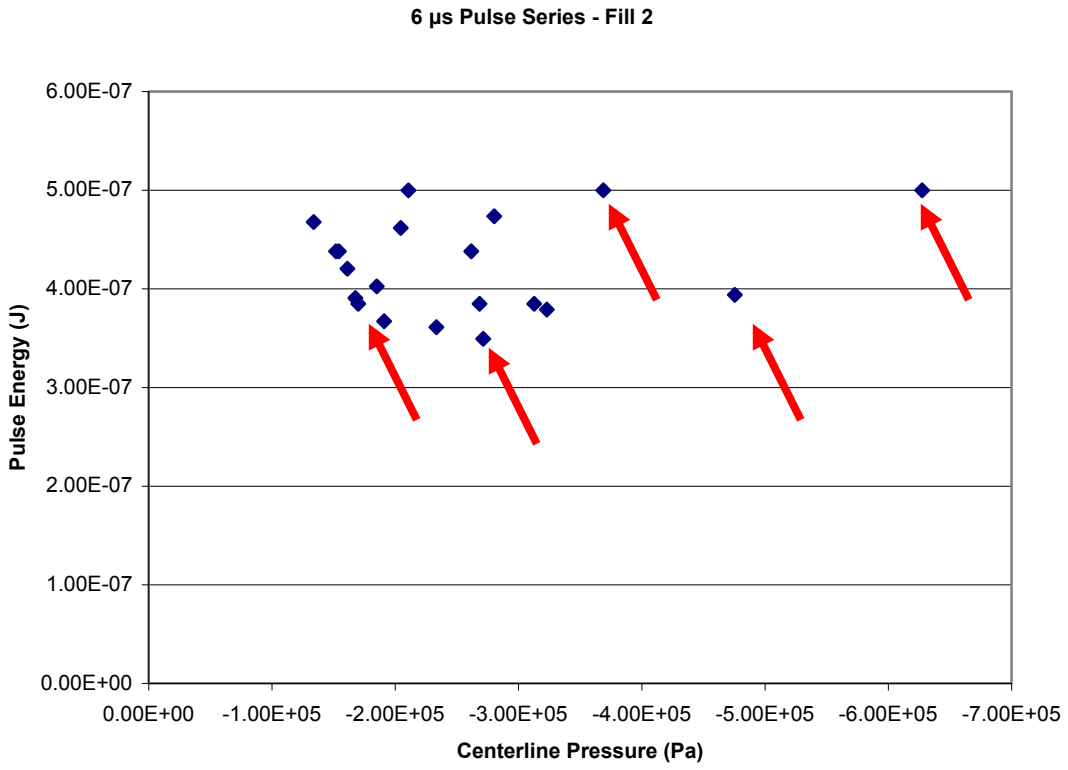

Figure 35: Fill 2 Data

Table 4 lists the data collected in this sequence, and it is presented graphically in Figure 35 with all problem points included; each of the questionable points has a red arrow pointing to it in the figure. Once again, there is a significant amount of noise in the gathered data which may interfere with the determination of a trend. However, the early values appear to hint at one, trending towards lower energy with decreasing pressure.

Table 4: Data from the Second Pulse Series

Test #	Fill Diameter (mm)	Power (mW)	Pulse Duration (μs)	Speed (rps)	Centerline Pressure (Pa)	Pulse Energy (J)	Note
1	37.44	79	5.92	208.18	-1.34E+05	4.68E-07	
2	38.14	74	5.92	213.05	-1.54E+05	4.38E-07	
3	38.44	74	5.92	210.43	-1.52E+05	4.38E-07	
4	38.93	71	5.92	211.5	-1.61E+05	4.20E-07	
5	39.41	65	5.92	212.45	-1.70E+05	3.85E-07	Very first shot
6	39.68	66	5.92	210.05	-1.68E+05	3.91E-07	
7	40.51	68	5.92	212.29	-1.85E+05	4.03E-07	
8	41.27	62	5.92	210.55	-1.91E+05	3.67E-07	
9	41.91	78	5.92	212.09	-2.05E+05	4.62E-07	
10	42.54	85	5.88	211.06	-2.11E+05	5.00E-07	
11	44	61	5.92	211.32	-2.33E+05	3.61E-07	
12	45.66	65	5.92	214.03	-2.68E+05	3.85E-07	
13	46	74	5.92	210.51	-2.62E+05	4.38E-07	
14	46.8	59	5.92	209.66	-2.71E+05	3.49E-07	3rd shot at first power
15	47.53	80	5.92	208.91	-2.80E+05	4.74E-07	
16	48.44	65	5.92	213.51	-3.13E+05	3.85E-07	
17	49.57	64	5.92	211.22	-3.23E+05	3.79E-07	
18	51.99	85	5.88	211.96	-3.69E+05	5.00E-07	Possible seed depletion
19	58.01	67	5.88	210.42	-4.75E+05	3.94E-07	Possible seed depletion
20	64.89	85	5.88	211.45	-6.28E+05	5.00E-07	Possible seed depletion

3.5.4. Third Fill

The third fill was used for a series of 6-μs pulses at 180 rps. In all, 4 threshold tests were completed before the fill was depleted to a level at which it could not ramp up to speed without the formation of a vapor column. However, the first test ramped all the way to the limit of the laser's power without forming a vapor column; as a result, some of the liquid was removed to start the next ramp at significantly increased tension. These tests appear to have experienced relatively violent transients, as the other 180 rps tests did not seem to remove as much fluid from the test section as these tests did in the 54 to 57 mm fill range. The results are given in Table 5.

Table 5: Data from the Third Pulse Series

Test #	Fill Diameter (mm)	Power (mW)	Pulse Duration (μs)	Speed (rps)	Centerline Pressure (Pa)	Pulse Energy (J)	Note
1	41.24	(none)					Liquid removed afterward
2	54.66	82	5.96	181.06	-2.78E+05	4.887E-07	
3	56.47	82	5.96	182.88	-3.12E+05	4.887E-07	
4	61.63	80	5.96	180.8	-3.79E+05	4.768E-07	

The data given in Table 5 is presented graphically in Figure 36. While there does appear to be a slight downward trend, there are only three points. Given the noise experienced by the other tests, it is surprising that there was so much consistency in this series. However, there are too few points to make a good determination of a trend with any confidence from these points alone.

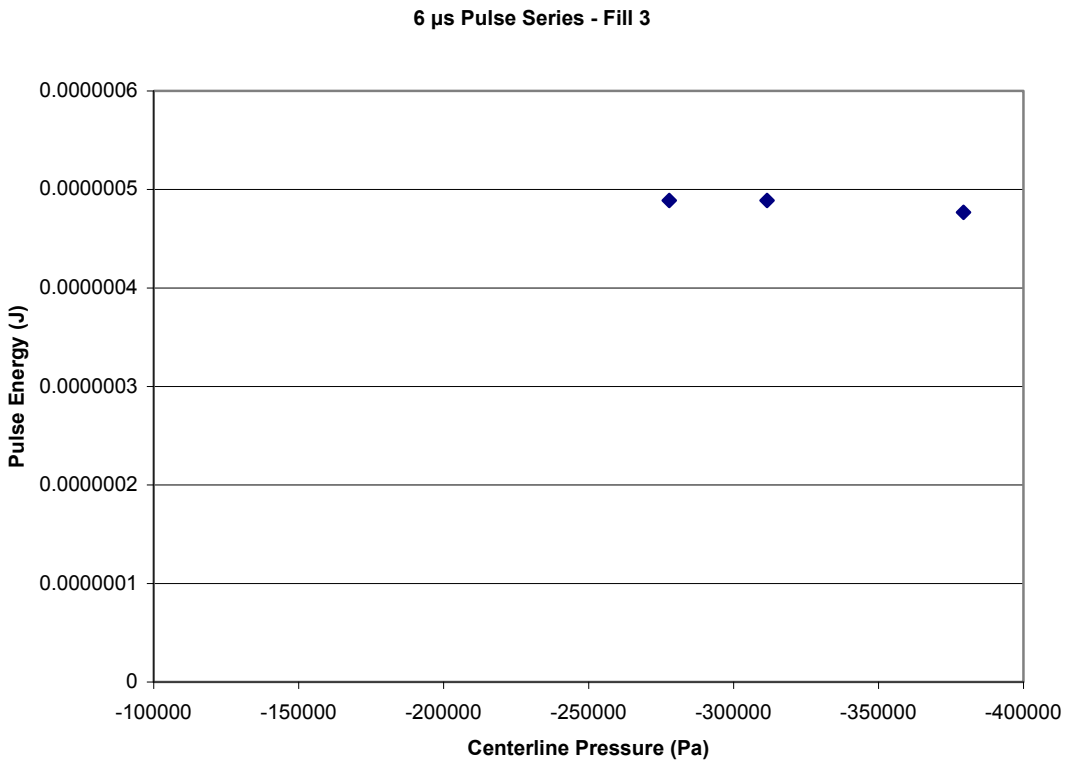

Figure 36: Fill 3 Chart

3.5.5. Fourth Fill

The fourth fill was used for a series of 6-µs pulses at 180 rps. In all, 19 threshold tests were completed before the fill was depleted to a level at which it could not ramp up to speed without the formation of a vapor column. However, the last test ramped all the way to the limit of the laser's power without forming a vapor column. In addition, the three tests previous to it had questionable qualities. All the data can be seen in Table 6; as it was the second fill of the day, it did not begin with test one. The third fill was performed immediately prior to this one.

Table 6: Data from the Fourth Pulse Series

Test #	Fill Diameter (mm)	Power (mW)	Pulse Duration (μs)	Speed (rps)	Centerline Pressure (Pa)	Pulse Energy (J)	Note
5	46.71	94	5.96	181.69	-1.78E+05	5.602E-07	
6	47.21	82	5.96	182.09	-1.85E+05	4.887E-07	
7	47.82	96	5.96	180.27	-1.86E+05	5.722E-07	
8	48.46	83	5.92	180.47	-1.95E+05	4.914E-07	
9	49.26	82	5.92	182.08	-2.10E+05	4.854E-07	
10	49.51	84	5.92	179.91	-2.06E+05	4.973E-07	
11	49.9	76	5.92	180.21	-2.12E+05	4.499E-07	
12	50.48	95	5.92	178.93	-2.15E+05	5.624E-07	
13	51.2	78	5.92	182.21	-2.36E+05	4.618E-07	
14	52.07	95	5.92	181.72	-2.45E+05	5.624E-07	
15	52.85	84	5.92	180.67	-2.52E+05	4.973E-07	
16	53.39	82	5.92	179.46	-2.54E+05	4.854E-07	
17	53.76	73	5.92	180.81	-2.64E+05	4.322E-07	
18	54.24	82	5.92	180	-2.68E+05	4.854E-07	
19	55.05	75	5.92	182.38	-2.89E+05	4.44E-07	
20	55.95	70	5.92	181.75	-2.99E+05	4.144E-07	2nd shot at first power
21	56.64	65	5.92	180.43	-3.03E+05	3.848E-07	3rd shot at first power
22	58.62	83	5.92	183.02	-3.44E+05	4.914E-07	Possible seed depletion
23	63.07	(none)					Possible seed depletion

The values from Table 6 are presented in Figure 37. Out of all the fill sequences, this one reveals the strongest trend. However, if all the questionable points are removed (indicated by red arrows in the figure), it becomes less apparent. In addition, the final test in the sequence did not end with the formation of a vapor column. This could be the

result of seed depletion. As the toner particles at this point had been experiencing significant centrifuging effects, it would not be surprising that enough would come out of solution to cause issues. It did appear that at the end of the test, some toner had caked out along the outermost points in the elbows of the test section, but it was not determined how much toner had actually undergone that.

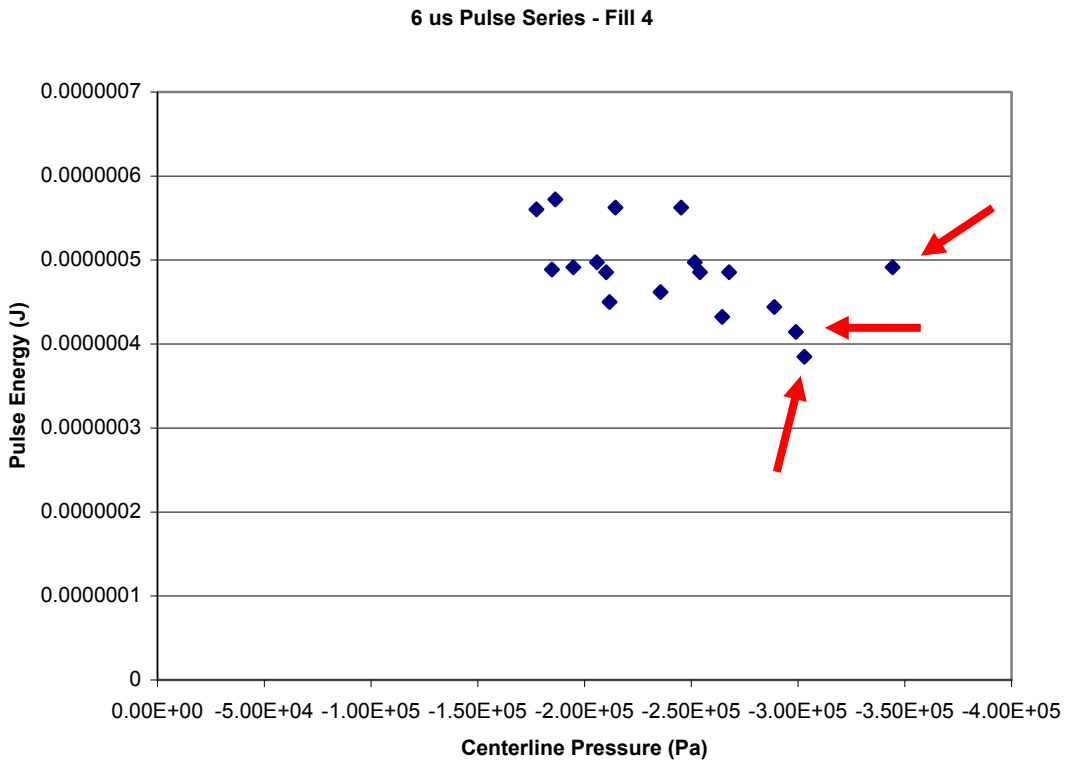

Figure 37: Fourth Fill Data

3.5.6. Discussion and Analysis

A collection of every test is charted in Figure 38 to compare the results. Each series is provided a different marker. By casual examination, it is clear that the 50

microsecond pulse data does not match the 6 microsecond pulse data at all. As there is a factor of more than 8 between their time scales, it is suspected that they are phenomenologically different. On the longer time scales, transient heat flow becomes more significant and may dominate the process. It is also possible that the motion of the seed particles comes into play in this time scale; they have more time to move into and out of the laser beam. In addition, since the total average pulse energy in the 50 microsecond tests is on the order of five times as great as that of the 6 microsecond tests, it is possible that the cavitation events may not be restricted to the zone of high power density near the focus point. They may happen elsewhere along the beam. This would only be expected to happen in the situation where seed particles are scarce, and the only way to deposit energy into the seed particles is to expand the applicable range. Such effects probably do play a role; it appears that seed depletion is an issue with many of the tests.

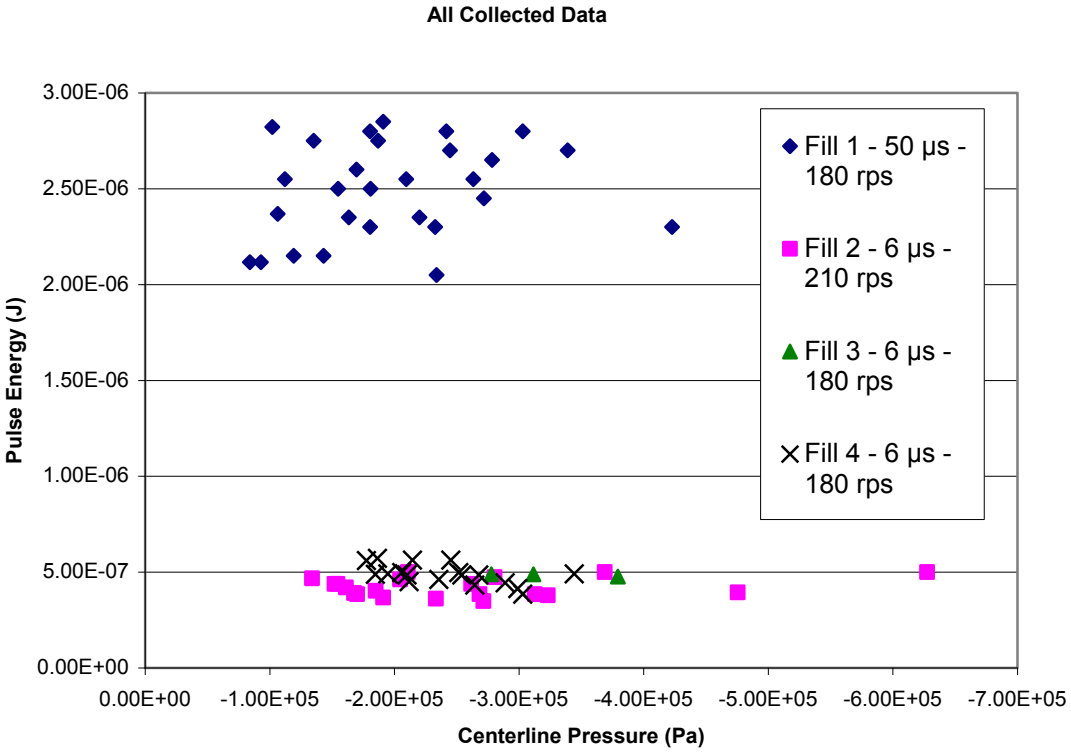

Figure 38: All Collected Data

When the 6-microsecond pulse data seen in Figure 38 appears without the 50-microsecond data, the differences between the tests becomes more pronounced. This is shown in Figure 39. Much of this may depend on variations in the initial fill; as particles may settle out of solution, even using the same solution at different times (as was done here) does not guarantee consistency between tests. In addition, the mounting is expected to be slightly different between tests. While care was taken to thread the test section onto the shaft consistently across fills, some variation is unavoidable.

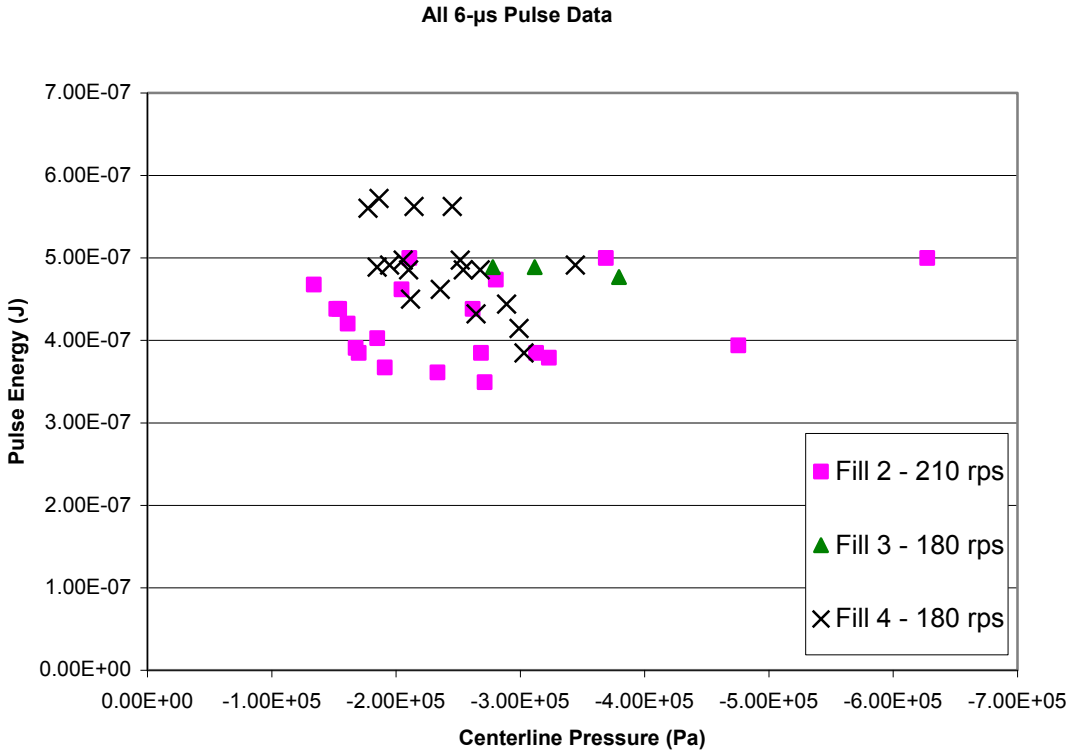

Figure 39: 6 µs Pulse Data

One bit of information that was largely neglected was the fluid temperature. While it was assumed that variations in it would not be significant, that may not be entirely true. The laser lab has limited air conditioning, and the containment box prevents most airflow. After prolonged periods of operation, the room warms up noticeably, and the air in the containment box seems much warmer. This should be no surprise; the rotary tool can use significant amounts of power and it is an enclosed space.

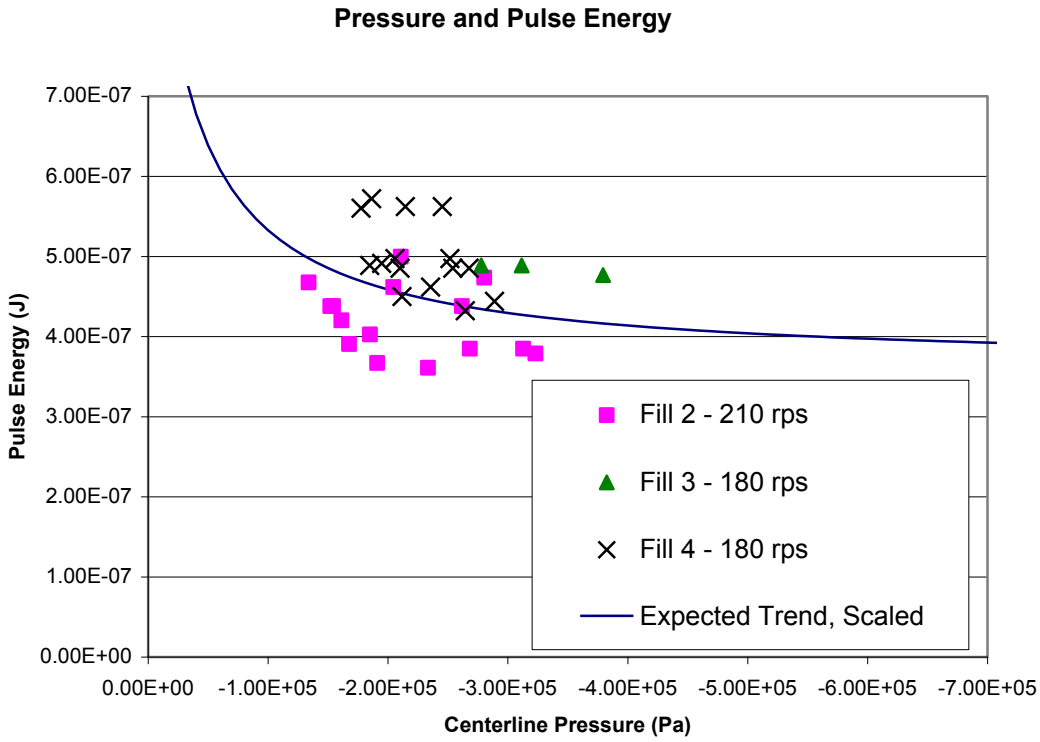

Figure 40: Comparison with Expected Trend

A comparison with the expected trend is shown in Figure 40 for the 6 microsecond pulse data. The questionable points were removed from the comparison, and the trend is that given by Eq. (54) using a scaling coefficient of 1.10E-7. It is not a curve fit or regression analysis; it is simply there for comparison purposes. The scatter in the data makes such a fit questionable at best; the results appear to neither confirm nor deny the expected trend. The 50 microsecond pulse data, however, appears to be essentially flat and would therefore tend to fit a scaled Eq. (52) rather than a scaled Eq. (54).

The comparison with the trend from Eq. (54) is important. It directly relates the experimental data to the energetics developed from the analysis of Bubble Theory. If future results show a poor fit to the trend, then Eq. (54) did not adequately capture the relevant phenomena. If future results do follow the trend, then the next step would be the development of the proper scaling coefficients. From there, one could determine the energy deposition characteristics that would be useful for full optimization of CTMFD systems.

4. COMPUTATIONAL FLUID DYNAMICS WORK

Computational Fluid Dynamics simulations can be very useful in the study of fluid systems. With advances in modern computing systems and CFD software, reasonable simulations can be performed on a wide variety of systems in decent amounts of time. In addition, modern tools allow the workload to be spread across several computer systems, reducing the computational demand on any one node. This allows for large, complex simulations to be performed even on commodity hardware.

CFD codes solve the equations of fluid mechanics in a discretized, numerical manner. An important step in a CFD simulation is the conversion of the flow geometry into a computational domain. In this process, a continuum of locations is divided into a finite number of places known as vertices, surfaces, and cells [32]. The flow conditions in each cell can then be described by a set number of parameters; the specific parameters in question are determined by the particular simulation. For example, in laminar simulations, turbulence parameters would not be included.

A CFD simulation needs more than just a mesh. Initial conditions are important, as are wall and boundary conditions. The entry/exit conditions need to be given, and the fluids themselves need to be defined (i.e., parameters like viscosity and density). In addition, the simulation models need to be chosen and their parameters defined. Turbulence models can be used, or flows can be forced into the laminar or inviscid regimes. If time dependence is not an issue, a steady-state simulation can be performed. Any applicable extra conditions -- motions in the reference frame, for instance, should

be determined as well. Stopping conditions such as time and convergence criteria should also be given.

When all the necessary conditions are provided and the simulation started, the solver steps through the calculations in an iterative manner. It applies the discretized forms of the fluids equations to the mesh and conditions, and solves them in a numerical rather than symbolic manner. Ideally, each iteration will rapidly bring the computed solution closer to the true one. Generally, a surrogate for the deviation from the true to the calculated solution is the residual in the computation (the difference between succeeding iterations; the exact meaning may vary from code to code). When the residual is very low, there is not a significant change in the computed solution from iteration to iteration, and the computed solution is said to have converged on that value. There may be several levels of convergence, not all of which may be accessible to the end user. If the calculation is transient, then the solver would proceed on to the next time step upon meeting the convergence criteria or iteration limits set by the user. If the residuals remain large, the solution is not converging and may instead diverge. That can quickly lead to nonphysical results and overflow errors, and should be avoided. Refinements to the mesh or other inputs can frequently be used to remedy such a situation.

4.1. CFD SIMULATION

The CFD simulation of a CTMFD was performed in CD-adapco ® Star-CCM+ 5.02.009, which uses finite volume computational techniques [32]. It ran on an ad-hoc Cluster of Workstations, each running a 64-bit version of Microsoft ® Windows™ (the

main node runs Windows Vista ®, while all the others run Windows™ 7) and the Win64 version of the MPICH2 v. 1.2.1p1 parallel computing library. The computers were connected via a Gigabit Ethernet network with Jumbo Frames enabled. The main node has a dual Intel ® Xeon™ 5520 CPU with a total of 8 physical cores and a total of 6 GB RAM. Each of the workers has 2 GB of RAM, but they have different processors. One has an Intel ® Core 2 Duo™ E7200, a second has an Intel ® Core 2 Duo™ E6700, and the third has a dual Xeon™ 3.2 (Nocona); the third node was largely used for offloading file storage.

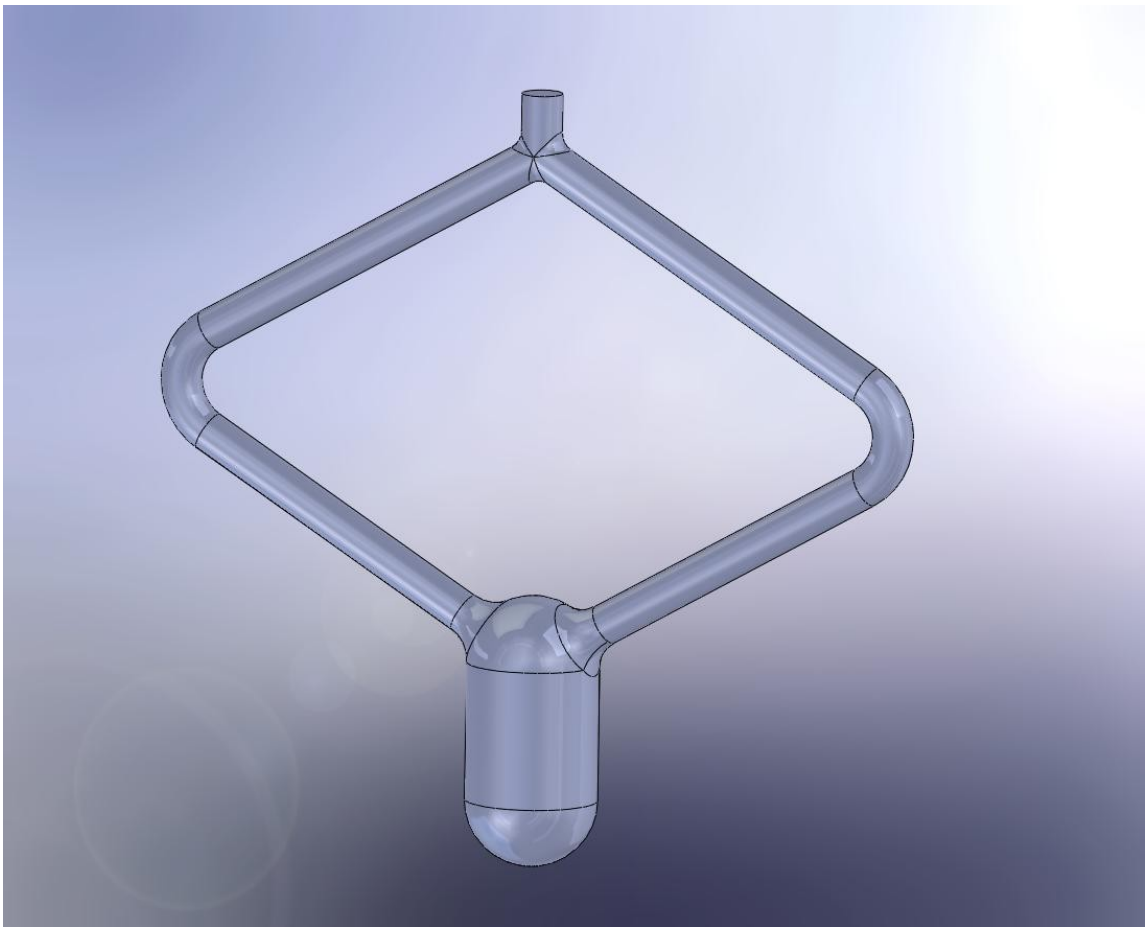

Figure 41: CAD model for CFD

The simulation used a model of a small CTMFD developed in SolidWorks™, exported in the IGES format, and imported into Star-CCM+ 4.02.007 for meshing. The CAD file describes the inner surface of the CTMFD glassware, and is depicted in Figure 41. It has a bulb with a total height of 26.5 mm and a diameter of 13 mm, an inner elbow radius of 71.056 mm, a tube inner diameter of 4 mm, and is shown in Figure 42.

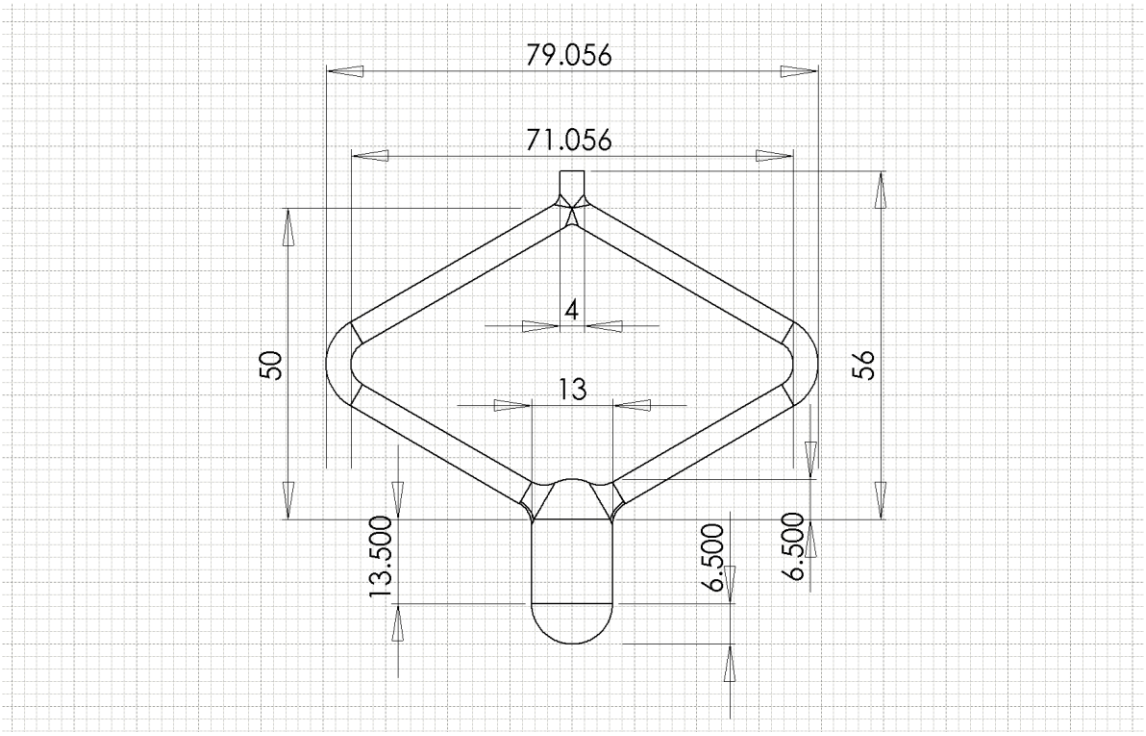

Figure 42: Model Dimensions (mm), Front

The straight parts of the arms in the CAD model are set at 60° angles from the centerline. Alternate views for the model can be seen in Figure 43 and Figure 44. They give the views from the top and side, respectively. Much of what they depict can be inferred from Figure 42.

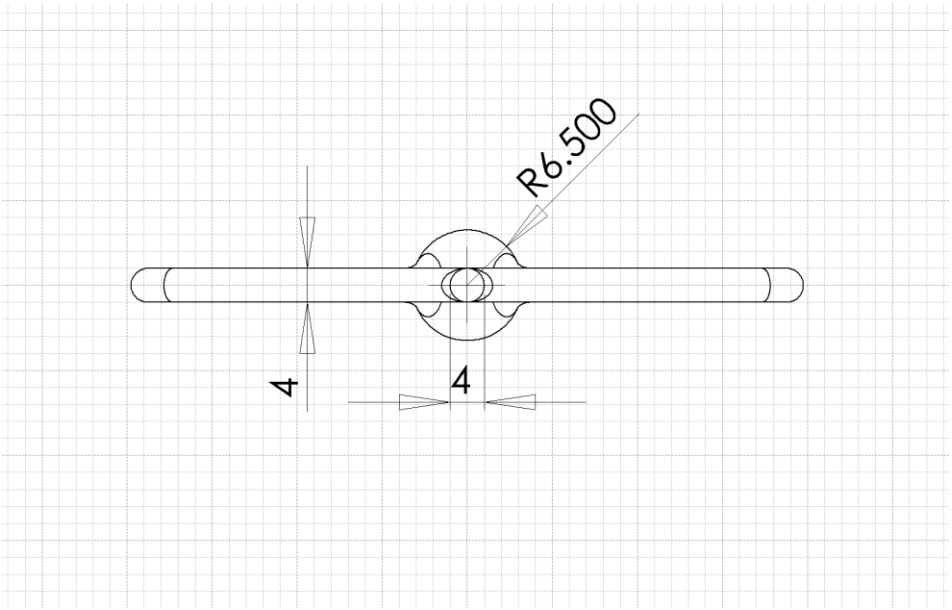

Figure 43: Model Dimensions (mm), Top

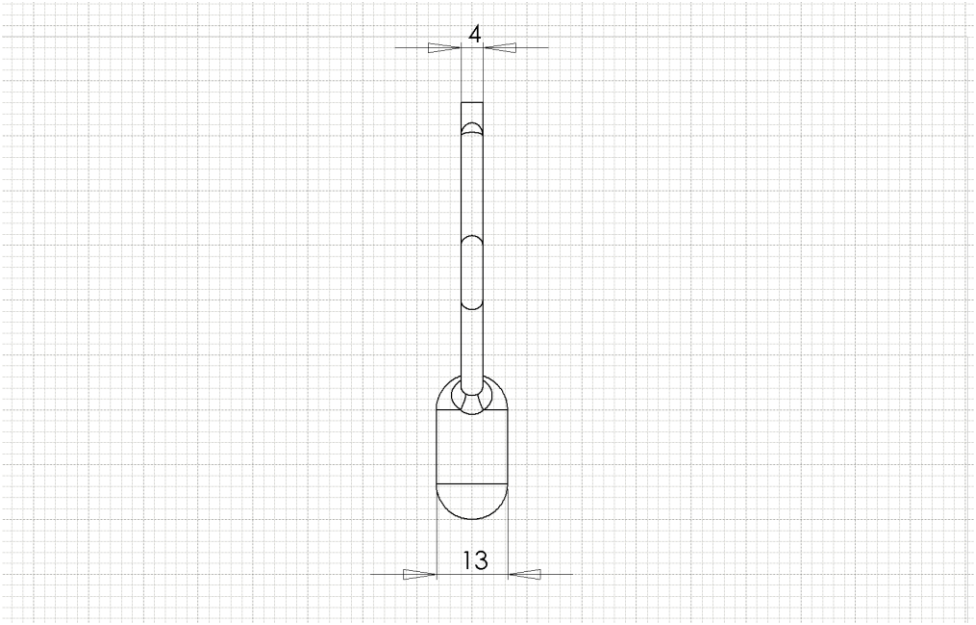

Figure 44: Model Dimensions (mm), Side

Meshing in Star-CCM+ 4.02.007 with the parameters given in Table 7 produced a geometry with 347,520 polyhedral cells, which can be seen in Figure 45. This file was

saved and eventually imported into Star-CCM+ 5.02.009, which has various improvements over earlier versions, especially in the realm of execution speed.

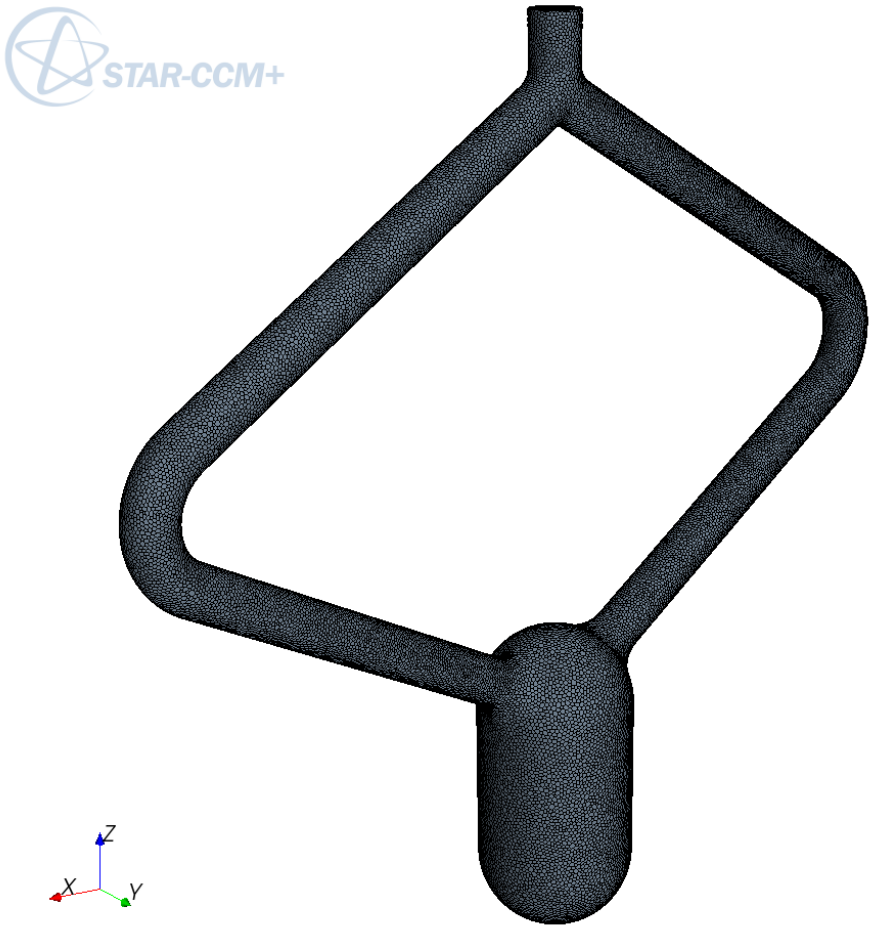

Figure 45: CFD Mesh

The simulation modeled a relatively simple transient case. For this simulation, phase changes and turbulence were disabled; flow was assumed to be laminar. The initial location of the air-acetone interface was placed at a radius of 2.5 cm by giving the

top of the arms in the CTMFD a fill fraction of 1 for air, and all other locations a fill fraction of 1 for acetone. This results in an initial fill similar to that presented in Figure 46, where "Phase 1" would be air. An air pressure boundary was chosen at the top, while the sides were set to be walls with the no-slip condition enforced.

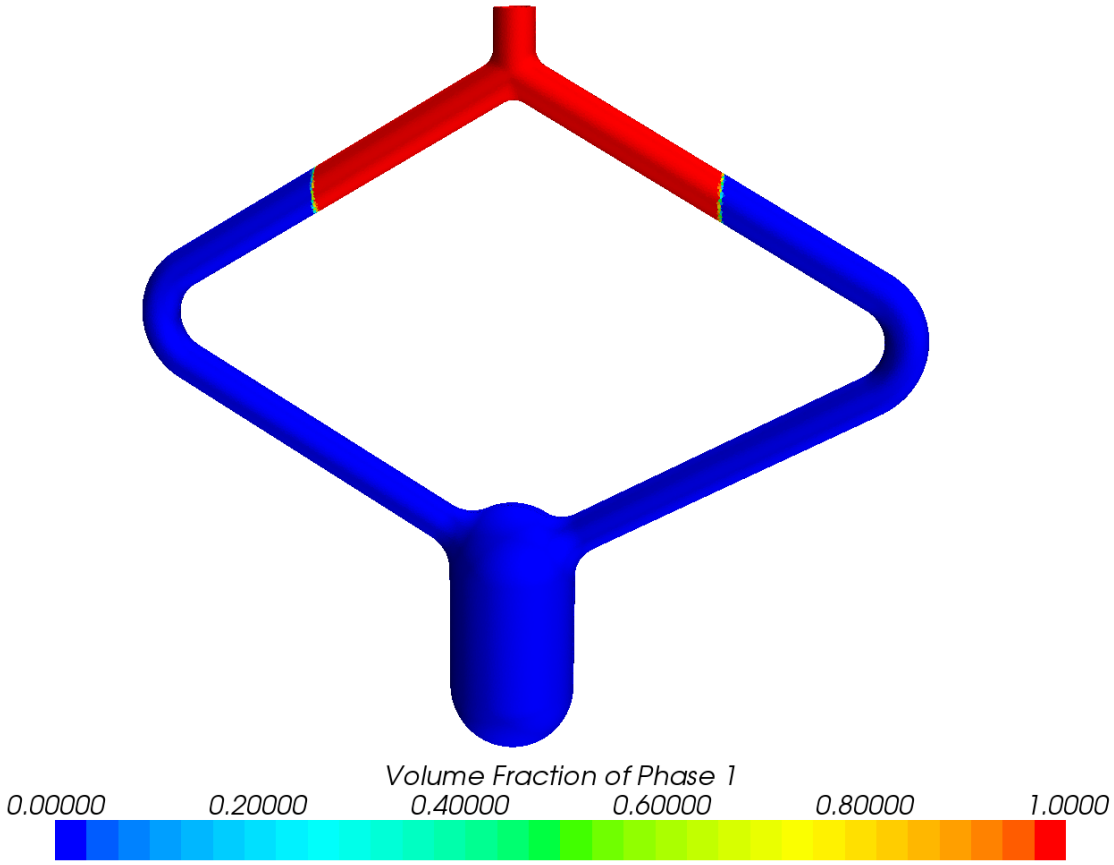

Figure 46: Initial Fill Example

The physics models are given in Table 8 and Table 9. This CFD model of a CTMFD serves as a framework for performing various CFD simulations. By changing

the relevant parameters and physics models, it can be used for more simulations than the one performed here.

The goal of the simulation was to simulate a simple ramp up from stationary to an operational speed, and to observe the time needed to reach equilibrium at the full, steady speed. Using a Java macro, the simulation settled for 0.2 s at the beginning, and then steadily ramped up in speed until it hit 180 rps at t=2.0 s. At that point, the CTMFD maintained a steady rotational rate. Output files were saved at intervals of 0.1 s of simulated time out to a maximum of 10.0 s, at which point the simulation ended.

Table 7: Meshing Information

Models	Surface Remesher Polyhedral Mesher Prism Layer Mesher
Reference Values	Base Size = 1 mm 4 Prism Layers Prism Layer Stretching Ratio = 1.5 Prism Layer Thickness = 33.3% of Base 36 Points/Circle Surface Growth Rate = 1.3 Surface Proximity: 2 Points in gap, 0 m search floor Relative Minimum Size = 25% of Base Relative Target Size = 35% of Base Polyhedral Density and Growth Factor both 1.0 Polyhedral Blending Factor = 1.0

The values given in Table 7 govern the meshing for this model in Star-CCM+. Polyhedral meshing was chosen because it has advantages over simpler mesh types (i.e., tetrahedral or trimmed); a model will need far fewer polyhedral cells for similar results.

The base size defines the size of cells in the coarse mesh, and the value of 1 mm was thought to provide an adequate resultant mesh; cells may be smaller as the need arises from geometric concerns, etc. The surface remesher was activated to improve the mesh quality and conformance to the original CAD model, and the prism layer mesher was enabled to improve the quality of turbulence modeling, if enabled in the future. It creates small layers of cells near the surfaces of the geometry [32].

The physics models are given in Table 8. Since the simulation is a three-dimensional, time-dependent, multiphase problem, the relevant models for that must be enabled. Laminar flow was assumed. The Eulerian Multiphase model used in conjunction with the Volume of Fluid model is for immiscible fluids where the mesh can resolve their interfaces [32], and requires the Segregated Flow models. Surface tension modeling was enabled, as was gravity and a rotating reference frame. The two fluids that were modeled were air and acetone, both modeled as incompressible using their built-in fluid characteristics. The surface tension value for acetone was not built-in, and was defined as 0.02272 N/m [21]. The initial distribution of air and acetone was defined by two field functions, inair (giving the volume fraction of air in a cell) and inwater (giving the volume fraction of acetone in a cell); both are given in Table 9.

Table 8: Physics Conditions

Models	Cell Quality Remediation Eulerian Multiphase Flow • Air, constant density = 1.18415 kg/m^3 • Acetone, constant density = 786.741 kg/m^3 Gravity Implicit Unsteady Laminar Multiphase EOS Multiphase Mixtures • Volume-Weighted Viscosity and Thermal Conductivity • Mass-Weighted Specific Heat Segregated Flow (2nd order convection) Segregated Fluid Isothermal, continuum temperature is 300 K Surface Tension Three Dimensional Volume of Fluid Model, 2nd order convection
Reference Values	Gravity: -9.81 m/s^2 in the z-direction Altitude=0 Valid Temperatures between 100 K and 5000 K Ref. Pressure = 101,325 Pa
Initial Conditions	Pressure = 101,325 Pa Temperature = 300 K Stationary Volume Fraction based on inair and inwater field functions

Table 9: Other Conditions

Solvers	Implicit Unsteady Timestep = 2.0E-4 s, 2nd order temporal discretization Rigid Body Motion defaults Segregated Flow defaults Segregated VOF defaults Segregated Energy defaults
Stopping Criteria	Maximum Inner Iterations = 15 10 s physical time
Reports: CurrentTime	Dimensions: Time Definition: $Time Units: s
Field function inair	dimensionless scalar defined as (($$Centroid[0])*($$Centroid[0])+($$Centroid[1])*($$Centroid[1])) < 0.000625) ? (($$Centroid[2] > 0.025) ? 1 : 0) : 0
Field function inwater	dimensionless scalar defined as ($inair > 0) ? 0 : 1

In Table 9, there are various parameters used in the CFD simulation that were not shown in Table 7 or Table 8. It includes the definitions of the inair and inwater field functions that determine the initial fluid distribution, as well as the CurrentTime report used by the Java macro to help conduct the simulation. The solvers used by the enabled models largely used their default parameters (relaxation factors, etc.). The Implicit Unsteady solver used a 2nd order temporal discretization scheme with a 0.2 ms timestep; this gives 12.96° of rotation per timestep at the full 180 rps. The maximum number of inner iterations in the calculations was 15. Those parameters resulted in residuals on the order of 10^{-3} to 10^{-4}, and the full 10 seconds of simulated time took nearly two weeks to compute across 11 CPU cores.

The important datum is the centerline pressure in the bulb. This value was extracted from three axial locations in the centerline of the bulb: one at the top of the cylindrical section where the upper dome begins, one 6.75 mm below the top of the cylindrical section, placing it dead center in the bulb, and one 13.5 mm below the top of the cylindrical section, at the bottom where it meets the lower dome. As the simulation was run, its state was saved at 0.1 s (simulated time) intervals for later data extraction; a macro was executed on each save file to help extract the data. Currently, only the mid-level (6.75 mm) point is used for analysis.

The centerline pressure values were examined over the time period of interest, and a determination was made as to how long it takes the pressure to settle down to a roughly constant value. This does not mean that the flows have themselves settled down, only that they have settled enough for the centerline pressure to reach its steady state value.

Both of the Java macros used in this effort (one to run the simulation, another to assist with data extraction) are given in Appendix C.

4.2. CFD RESULTS

Using a macro for assistance, the relevant values were captured from the simulation after completion. The simulation was operated by another Java macro to adjust the rotation rate as a function of simulated time; the main User Interface in Star-CCM+ was not otherwise amenable to fluctuating rotation rates.

The extracted values for the pressure at the midpoint are given in Table 10 for the period before and during the speed ramp-up. It lists the calculated pressure as well as

the drop from the reference (atmospheric) to the centerline pressure; both the CFD results and the results of the application of Eq. (42) are shown for comparison.

Table 10: CFD Ramp-Up

Time	Rotation Rate	CFD Pressure	ΔP CFD	Pressure Eq. (42)	ΔP Eq. (42)
0	0	1.01E+05	0.00E+00	1.01E+05	0.00E+00
0.1	0	1.02E+05	-4.00E+02	1.01E+05	0.00E+00
0.2	0	1.02E+05	-4.00E+02	1.01E+05	0.00E+00
0.3002	10	1.01E+05	5.29E+02	1.00E+05	9.71E+02
0.4002	20	9.80E+04	3.28E+03	9.74E+04	3.88E+03
0.5002	30	9.34E+04	7.89E+03	9.26E+04	8.74E+03
0.6002	40	8.70E+04	1.43E+04	8.58E+04	1.55E+04
0.7002	50	7.86E+04	2.27E+04	7.71E+04	2.43E+04
0.8002	60	6.84E+04	3.29E+04	6.64E+04	3.49E+04
0.9002	70	5.63E+04	4.51E+04	5.38E+04	4.76E+04
1.0002	80	4.22E+04	5.91E+04	3.92E+04	6.21E+04
1.1002	90	2.63E+04	7.50E+04	2.27E+04	7.86E+04
1.2002	100	8.48E+03	9.28E+04	4.26E+03	9.71E+04
1.3002	110	-1.12E+04	1.13E+05	-1.61E+04	1.17E+05
1.4002	120	-3.27E+04	1.34E+05	-3.84E+04	1.40E+05
1.5002	130	-5.60E+04	1.57E+05	-6.27E+04	1.64E+05
1.6002	140	-8.11E+04	1.82E+05	-8.89E+04	1.90E+05
1.7002	150	-1.08E+05	2.09E+05	-1.17E+05	2.18E+05
1.8002	160	-1.37E+05	2.38E+05	-1.47E+05	2.48E+05
1.9002	170	-1.67E+05	2.68E+05	-1.79E+05	2.81E+05
2.0002	180	-1.99E+05	3.01E+05	-2.13E+05	3.14E+05

The ramp-up period was from 0.2 s to 2.0 s; that period is contained in Table 10. After that, the rotation rate was held constant and the solution was allowed to settle on equilibrium values. The results for the beginning of the equilibration period, 2 s to 6 s, are given in Table 11. As can be seen, the values quickly converge to a steady-state solution.

Table 11: Post-Ramp Values

Time	Rotation Rate	CFD Pressure	ΔP CFD	Pressure Eq. (42)	ΔP Eq. (42)
2.1002	180	-2.01E+05	3.02E+05	-2.13E+05	3.14E+05
2.2002	180	-2.02E+05	3.03E+05	-2.13E+05	3.14E+05
2.3002	180	-2.03E+05	3.04E+05	-2.13E+05	3.14E+05
2.4002	180	-2.03E+05	3.04E+05	-2.13E+05	3.14E+05
2.5002	180	-2.03E+05	3.05E+05	-2.13E+05	3.14E+05
2.6002	180	-2.03E+05	3.05E+05	-2.13E+05	3.14E+05
2.7002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
2.8002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
2.9002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.0002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.1002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.2002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.3002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.4002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.5002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.6002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.7002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.8002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
3.9002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.0002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.1002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.2002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.3002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.4002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.5002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.6002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.7002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.8002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
4.9002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.0002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.1002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.2002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.3002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.4002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.5002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.6002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.7002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.8002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
5.9002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
6.0002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05

Table 12: Long-Term Values

Time	Rotation Rate	CFD Pressure	ΔP CFD	Pressure Eq. (42)	ΔP Eq. (42)
6.1002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
6.2002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
6.3002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
6.4002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
6.5002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
6.6002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
6.7002	180	-2.04E+05	3.05E+05	-2.13E+05	3.14E+05
6.8002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
6.9002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.0002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.1002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.2002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.3002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.4002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.5002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.6002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.7002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.8002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
7.9002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.0002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.1002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.2002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.3002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
8.4002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.5002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.6002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
8.7002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
8.8002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
8.9002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
9.0002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
9.1002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
9.2002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
9.3002	180	-2.04E+05	3.06E+05	-2.13E+05	3.14E+05
9.4002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
9.5002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
9.6002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
9.7002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
9.8002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
9.9002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05
10.0002	180	-2.05E+05	3.06E+05	-2.13E+05	3.14E+05

The post-ramp values given in Table 11 do not end at the end of the simulation; that occurs at 10 s. Continuing on to the final value, Table 12 gives the long-term approach in the results. The deviations from steady-state tend to be small. After 10 s, it was assumed that any further changes in the centerline pressure would be negligible, and that it would represent a steady-state condition.

The values given in Table 10, Table 11, and Table 12 can be charted to graphically show the time dependence of the difference between atmospheric and CTMFD centerline pressures. The comparison to values extracted from Eq. (42) are shown in Figure 47 and Figure 48.

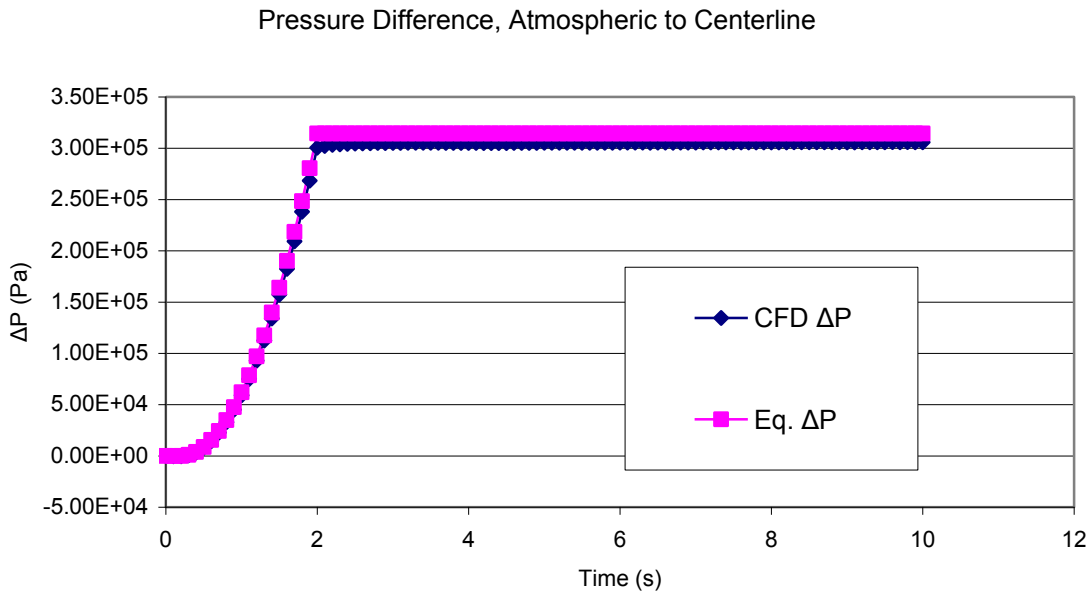

Figure 47: CFD Pressure Drop

The pressure differences determined by the simple CFD simulation and the equation, as shown in Figure 47, are very close to each other. The equation assumed that the rotation rate was steady, not fluctuating, and yet gave remarkably close values to the CFD calculations. To make this clearer, one can examine Figure 48. It gives the pressure determined by the CFD calculations as a fraction of its steady-state (10 s) value and as a fraction of the pressure drop determined by Eq. (42). Even during the ramp period, the CFD-determined pressure drop closely followed the pressure drop from Eq. (42); outside of the very early stages of the ramp, it was well within 10% of the equation's value.

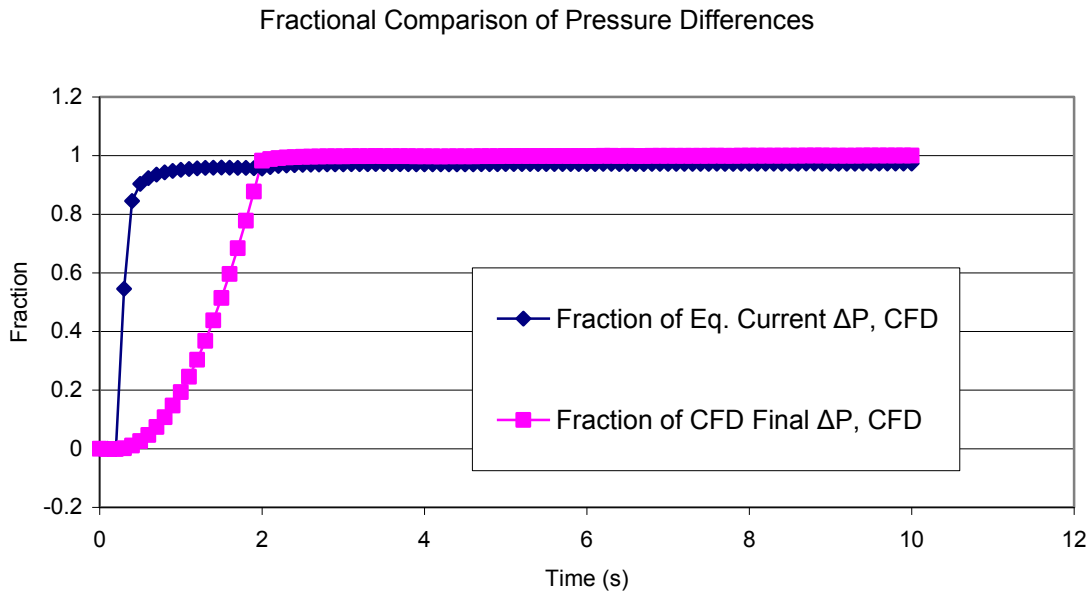

Figure 48: CFD ΔP Fractions

The fractional comparison is given in Figure 48. The figure shows two views of the CFD-determined centerline pressure difference from atmospheric. One is the fraction of the pressure difference as determined by Eq. (42) at the given time. The other is the fraction of the pressure difference at the given time compared to the long-term (10 s) value. From the figure, it can be seen that the centerline pressure in the CFD calculation approaches its steady-state value well within one second after the end of the ramp. It also shows that the centerline pressure during ramp-up closely tracks the values given by Eq. (42) for the speed at the given time. The steady-state values for the lab-frame velocity magnitude and pressure field are shown in Figure 49 and Figure 50; they show the distribution of those values using cutaway views of the simulated test section.

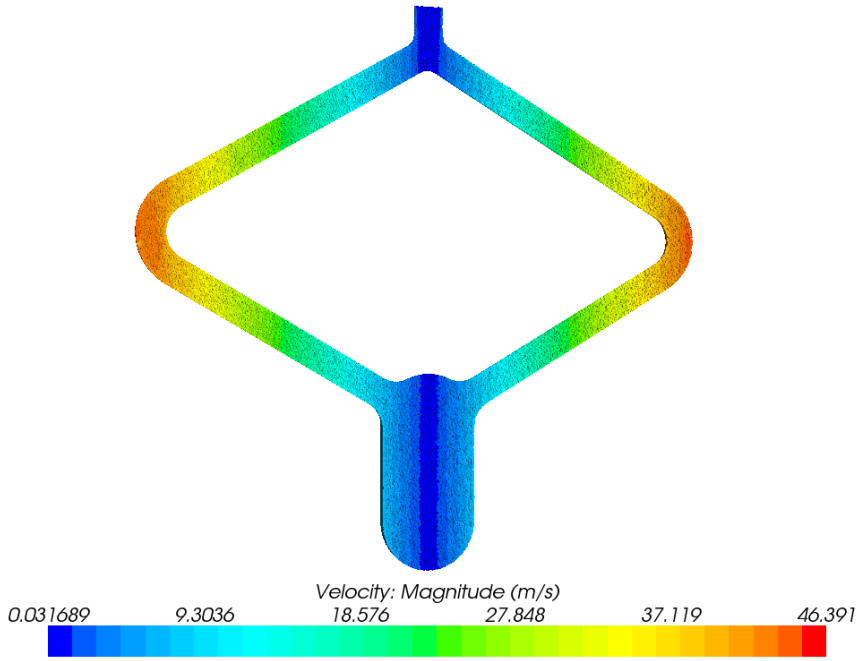

Figure 49: Lab-Frame Velocity Distribution

In Figure 49, a cutaway view of the CFD model can be seen with the steady-state velocity distribution. It is presented in the lab frame; within the rotating reference frame, the velocity magnitudes are close to zero, as is expected. A similar cutaway view is presented in Figure 50 for the pressure field. It shows a relatively constant pressure in the upper arms; this region is filled with low-density air, so the result is believable. Below that, the relatively high-density acetone shows a large radial pressure gradient, which is steeper going out to toward the elbows. Near the center, the gradient is much milder.

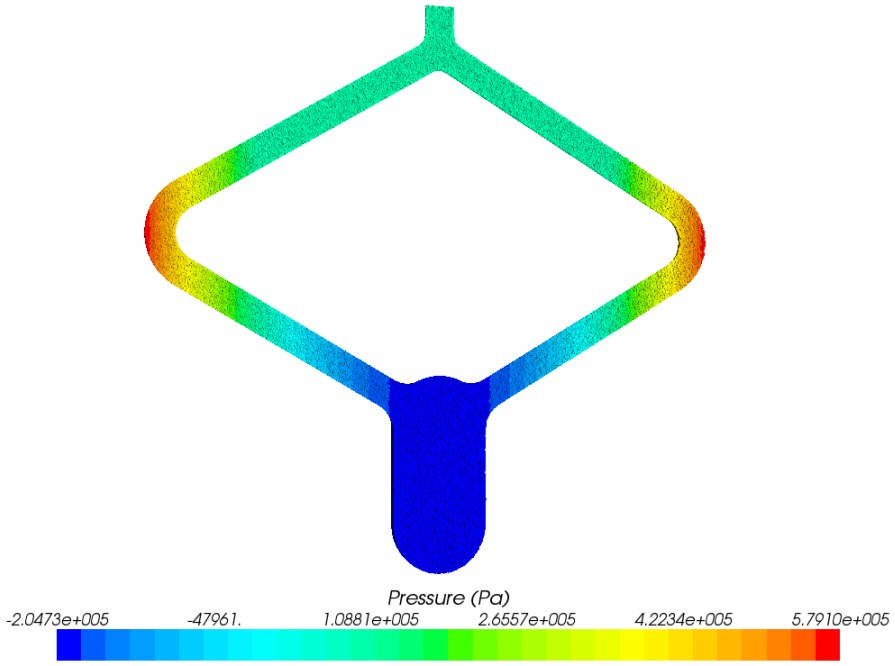

Figure 50: Pressure Distribution

It would appear from the simulation that the air density makes a slight difference as to the final centerline pressure. The difference between the simple calculation given

in Eq. (42) and the simulation is on the order of 2-3% at steady state, using ambient pressure as a reference.

It is also notable that the pressures immediately after the speed has finished ramping up to its maximum value are at 98.3% of their steady state value. After just 0.2 s post-ramp, the pressures cross the 99% mark, reaching 99.2% of their steady-state values. At 0.4 s post-ramp, the pressures are at 99.5% of the steady-state value. They slowly, with minor oscillations, approach the full steady-state value after that. At 8.0 s after the speed has finished ramping up (10.0 s in the simulation) from 0 to 180 rps (in 1.8 s), the pressures are considered to be at their steady-state values and these numbers are used for the earlier comparisons.

5. CONCLUSIONS

This research was intended to increase the current knowledge of the behavior of CTMFD systems. A multi-pronged approach was taken; this research applied theoretical, computational, and experimental techniques to combat the lack of knowledge

The first major goal was to design, construct, and operate an experimental facility for CTMFD system evaluation. Such a facility was successfully built, and it is operational.

A second goal was to explore the use of laser-induced cavitation as an experimental technique in general, with an emphasis on CTMFD systems. This appears to be a valid technique, as the experimental operations were successful at achieving laser-induced cavitation in a seeded fluid.

The third goal involved exploring the energetics of a CTMFD system. The application of theory suggested the result would be a curve based on the pressure in the fluid. While the experiment as-is successfully determines the broad range of energies that may work, the results are currently too noisy to confidently establish curve parameters.

The final major goal was the development of a CFD framework for CTMFD modeling. This was achieved; a trial run has provided an insight into the rapid approach to steady-state pressures a CTMFD may experience.

While the data collected in the current experimental runs are too noisy for the proper development of an empirical model, they do suggest that future runs with improved techniques would be able to achieve that goal. In addition, the CFD framework developed here can be used to cross-compare experimental results with computer simulations as well as with the theoretical models developed here.

5.1. KEY FINDINGS

The experiment and simulation demonstrated the ability of the facilities to test CTMFD systems and the potential to extract their operational characteristics. The experiment showed a certain viability for the technique of laser-induced cavitation in a seeded fluid, and demonstrated some of the associated limitations as well. Some of the key conclusions are listed:

- CTMFD pressures come to their steady-state values relatively fast, according to a CFD simulation
- Triggering a CTMFD has critical phenomena that occur on less than ms timescales; the growth of small bubbles to the full vapor column is fast and violent
- There is a clear threshold for the time-dependence on the energy deposition rate; if it happens too slowly, then it is phenomenologically different from a much faster or instant deposition

5.2. FUTURE WORK

There are certain issues that need to be resolved before quantitative results can be obtained from this research:

- Resolution of the issues with glassware curvature at the laser entrance; the optically flat glassware should be used in future tests
- Full characterization of the seeding material, including particle size distributions, time to settle out of solution, opacity, thermal conductivity, heat capacity, interface energy with the surrounding fluid, etc.
- Reduction of the vibrations in the experiment, which may have large influences on the outcomes
- Finding a more effective way to clean the test sections
- General reduction in the noise/scatter of the data
- Further CFD simulations to compare with theory and experimental data

In addition, the systems developed can be used to collect data for other fluids and conditions, not just acetone near room temperature. Dyes may be explored as opposed to particulate seeds. Finally, tests with radioactive sources should be conducted as well to characterize the system's responses to different types of radiation.

REFERENCES

[1] Trevena, D. H. *Cavitation and Tension in Liquids*. Bristol, UK: IOP Publishing, 1987.

[2] Briggs, Lyman J. *The Limiting Negative Pressure of Acetic Acid, Benzene, Aniline, Carbon Tetrachloride, and Chloroform.* Journal of Chemical Physics 19.7 (1951): 970-972.

[3] Zheng, Q., D. J. Durben, G. H. Wolf, C. A. Angell. *Liquids at Large Negative Pressures: Water at the Homogeneous Nucleation Limit*. Science 254.5033 (1991): 829-832.

[4] Scholander, P. F., H. T. Hammel, Edda D. Bradstreet, E. A. Hemmingsen. *Sap Pressure in Vascular Plants: Negative Pressure can be Measured in Plants*. Science 148.3668 (1965): 339-346.

[5] Taleyarkhan, Rusi, J. Lapinskas, Y. Xu. *Tensioned Metastable Fluids and Nanoscale Interactions with External Stimuli—Theoretical-cum-Experimental Assessments and Nuclear Engineering Applications*. Nuclear Engineering and Design 238.7 (2008): 1820-1827.

[6] Brennen, Christopher E. *Cavitation and Bubble Dynamics*. New York: Oxford University Press, Inc., 1995.

[7] Leighton, T. G. *The Acoustic Bubble*. London: Academic Press Limited, 1994.

[8] Neppiras, E. A. *Acoustic Cavitation*. Physics Reports 61.3 (1980): 159-251.

[9] Collier, John G., Thome, John R. *Convective Boiling and Condensation*. 3rd ed. New York: Oxford University Press, Inc., 1996.

[10] U.S. Food and Drug Administration. *Risk of Burns from Eruptions of Hot Water Overheated in Microwave Ovens*. Last Updated 25 April 2012, Retrieved 19 Oct 2012, http://www.fda.gov/Radiation-EmittingProducts/RadiationEmittingProductsandProcedures/HomeBusinessandEntertainment/ucm142506.htm

[11] Fisher, John C. *The Fracture of Liquids*. Journal of Applied Physics 19.11 (1948): 1062-1067.

[12] Lienhard, J.H., Karimi, A. *Homogeneous Nucleation and the Spinodal Line*. Journal of Heat Transfer 103.1 (1981): 61-64.

[13] Poole, Peter H., Francesco Sciortino, Ulrich Essmann, H. Eugene Stanley. *Spinodal of Liquid Water*. Physical Review E 48.5 (1993): 3799-3817

[14] Holbrook, N. Michele, Michael J. Burns, Christopher B. Field. *Negative Xylem Pressures in Plants: A Test of the Balancing Pressure Technique*. Science 270.5239 (1995): 1193-1194.

[15] Glaser, Donald A. *Some Effects of Ionizing Radiation on the Formation of Bubbles in Liquids*. Physical Review 87.4 (1952): 665.

[16] Seitz, Frederick. *On the Theory of the Bubble Chamber*. Physics of Fluids 1.1 (1958): 2-13.

[17] E. Behnke, J. I. Collar, P. S. Cooper, K. Crum, M. Crisler, M. Hu, I. Levine, D. Nakazawa, H. Nguyen, B. Odom, E. Ramberg, J. Rasmussen, N. Riley, A. Sonnenschein, M. Szydagis, R. Tschirhart. *Spin-Dependent WIMP Limits from a Bubble Chamber*. Science 319.5865 (2008): 933-936.

[18] Knoll, Glenn F. *Radiation Detection and Measurement*. 3rd ed. Hoboken, NJ: John Wiley & Sons, Inc., 2000.

[19] Bertolotti, M., D. Sette, F. Wanderlingh. *On the Possibility of High Energy Particle Detectors Based on Ultrasonic Cavitation*. Nuclear Instruments and Methods 35.1 (1965): 109-112.

[20] Briggs, Lyman J. *Limiting Negative Pressure of Water*. Journal of Applied Physics 21.7 (1950): 721-722.

[21] W. M. Haynes, ed. *CRC Handbook of Chemistry and Physics*. 93rd Edition (Internet Version 2013), CRC Press/Taylor and Francis, Boca Raton, FL.

[22] National Institute of Standards and Technology. *Acetone*. Retrieved 5 Oct 2012, http://webbook.nist.gov/cgi/cbook.cgi?ID=C67641&Units=SI&Mask=4#Thermo-Phase

[23] Sigma-Aldrich. *Carbon, Mesoporous Nanopowder, <500 nm Particle Size (DLS), >99.95% Trace Metals Basis*. CAS 1333-86-4. Retrieved 18 Oct 2012, http://www.sigmaaldrich.com/catalog/product/aldrich/699632?lang=en&region=US

[24] Huang, Jing, K. Padmanabhan, O. M. Collins. *The Sampling Theorem with Constant Amplitude Variable Width Pulses*, Circuits and Systems I: Regular Papers, IEEE Transactions on, 58.6 (2011): 1178-1190.

[25] Barr, Michael. *Pulse Width Modulation*, Embedded Systems Programming (2001): 103-104. Accessible at http://www.barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

[26] Ruilang Smarthome Limited, *Types of Dimmer*. Retrieved 19 Oct 2012, http://www.dimming.org/Technical/detail/705.html

[27] Axelson, Jan. *Serial Port Complete*. Madison, WI: Lakeview Research, 1998.

[28] Tissue, Bryan M. *Beer-Lambert Law*. Virginia Tech, 2003, Retrieved 19 Oct 2012, http://www.files.chem.vt.edu/chem-ed/spec/beerslaw.html

[29] Newport Corporation. *Gaussian Beam Optics Tutorial*. Retrieved 7 Oct 2012, http://www.newport.com/servicesupport/tutorials/default.aspx?id=112

[30] Coherent, Inc. *Coherent CUBE™ Laser System Operator's Manual*. Rev. AB. Santa Clara, CA: Coherent, 2005. Accessible at http://www.coherent.com/downloads/Cube_RevAB.pdf

[31] National Semiconductor Corporation. *LM556 Dual Timer*. Santa Clara, CA: National Semiconductor, March 2000. Texas Instruments Literature Number: SNAS549. Accessible at http://www.ti.com/lit/ds/symlink/lm556.pdf

[32] CD-adapco. *User Guide: Star-CCM+ Version 5.02*. Melville, NY: CD-adapco, 2009.

APPENDIX A

ELECTRIC CIRCUIT SCHEMATICS

This Appendix contains the electrical schematics for the circuits designed and built as part of this research effort.

A.1 CONTAINMENT BOX INTERLOCK

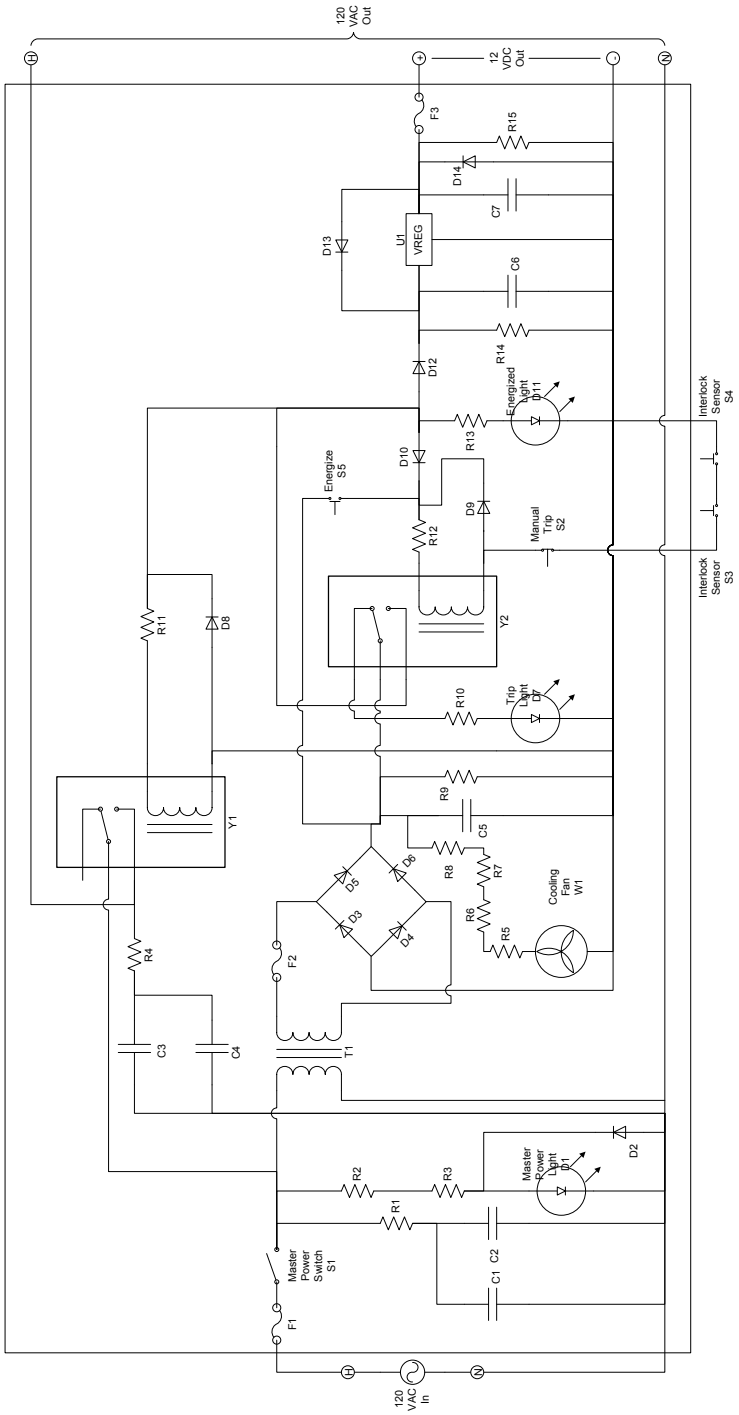

Figure 51: Containment Box Interlock Schematic

In Figure 51, the electrical schematic for the interlock on the containment box is given. The components labeled in the figure have values given in Table 13.

Table 13: Containment Box Interlock Components

C1: 0.047 µF	F1: 10 A	S1: R13-28D-06-R
C2: 0.047 µF	F2: 2.5 A	S2: GSW01-R
C3: 0.047 µF	F3: 1.5 A	S3: GPB006-WH-R
C4: 0.047 µF		S4: V3L-1465-D8
C5: 6800 µF	R1: 47 Ω	S5: V3L-1465-D8
C6: 3300 µF	R2: 4.7k Ω 5W	
C7: 1000 µF	R3: 4.7k Ω 5W	T1: QC-5126
	R4: 47 Ω	
D1: 5100H1	R5: 10 Ω	U1: L7812CV
D2: 1N5822G	R6: 10 Ω	
D3, D4, D5, D6: KBU6J	R7: 10 Ω	W1: 3110GL-B4W-B19-P53
D7: 5100H7	R8: 10 Ω	
D8: 1N5822G	R9: 10k Ω	Y1: K10P-11D15-12
D9: 1N5822G	R10: 750 Ω ½ W	Y2: LB2-12DS-R
D10: 1N5822G	R11: 33 Ω 5 W	
D11: 5102H5-5V	R12: 33 Ω 5 W	
D12: 80SQ045	R13: 1k Ω ½ W	
D13: 1N5822G	R14: 10k Ω	
D14: 1N5822G	R15: 10k Ω	

A.2 SPEED SENSOR

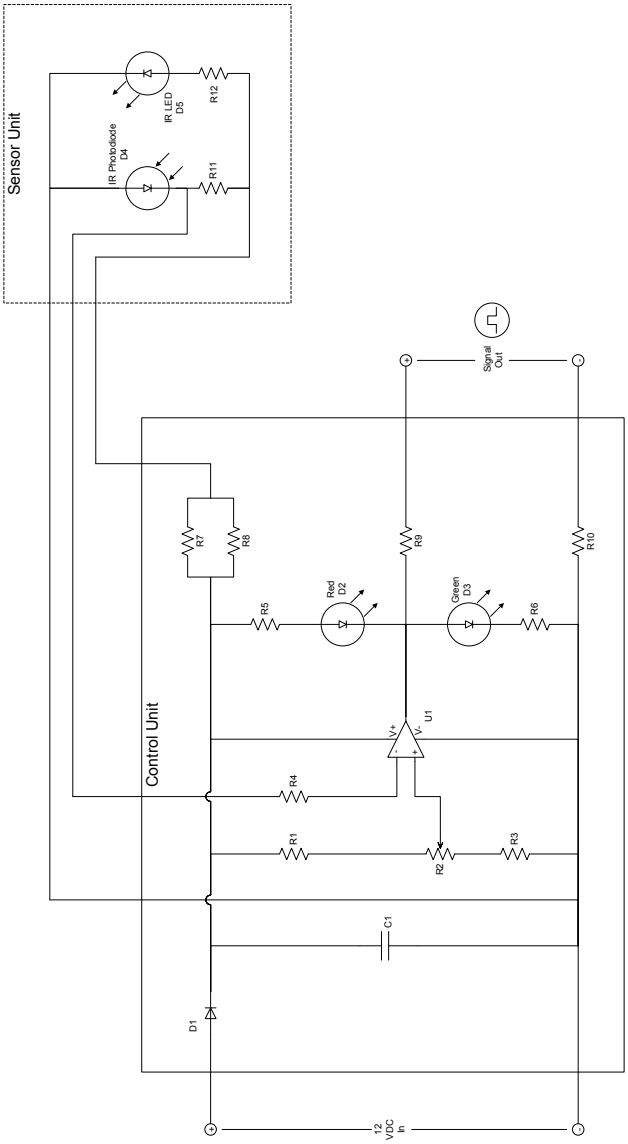

Figure 52: Speed Sensor Schematic

In Figure 52, the electrical schematic for the speed sensor circuit is shown. The components shown in it are detailed in Table 14.

Table 14: Speed Sensor Components

C1: 560 pF	R1: 470 Ω
	R2: 5k Ω
D1: 1N5822G	R3: 470 Ω
D2: HLMP-1700-B0002	R4: 220 Ω
D3: HLMP-1790-A0002	R5: 3.9k Ω
D4: BPV22NF(L)	R6: 3.9k Ω
D5: L-34F3CS	R7: 10 Ω
	R8: 10 Ω
U1: LF353N	R9: 220 Ω
	R10: 220 Ω
	R11: 1M Ω
	R12: 560 Ω

A.3 SPEED SENSOR ADAPTER

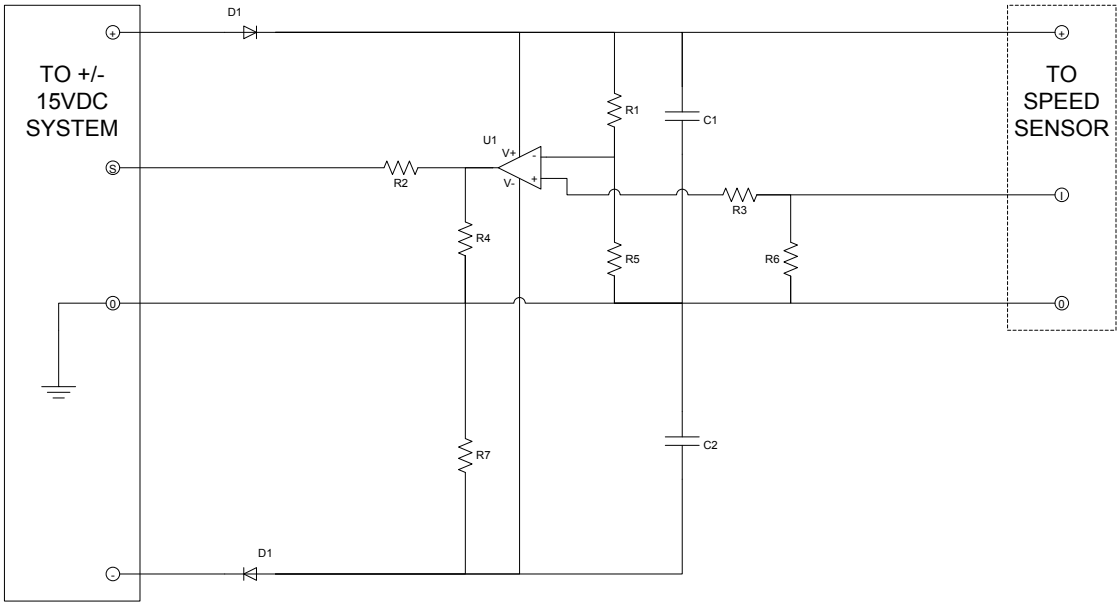

Figure 53: Speed Sensor Adapter Schematic

The electrical schematic for the speed sensor adapter is given in Figure 53. The corresponding component details are given in Table 15.

Table 15: Speed Sensor Adapter Components

C1: 10 pF	R1: 470k Ω
C2: 10 pF	R2: 220 Ω
	R3: 10k Ω
D1: 50SQ100	R4: 1M Ω
D2: 50SQ100	R5: 470k Ω
	R6: 1M Ω
U1: MC34071A	R7: 1M Ω

A.4 CAVITATION SENSOR

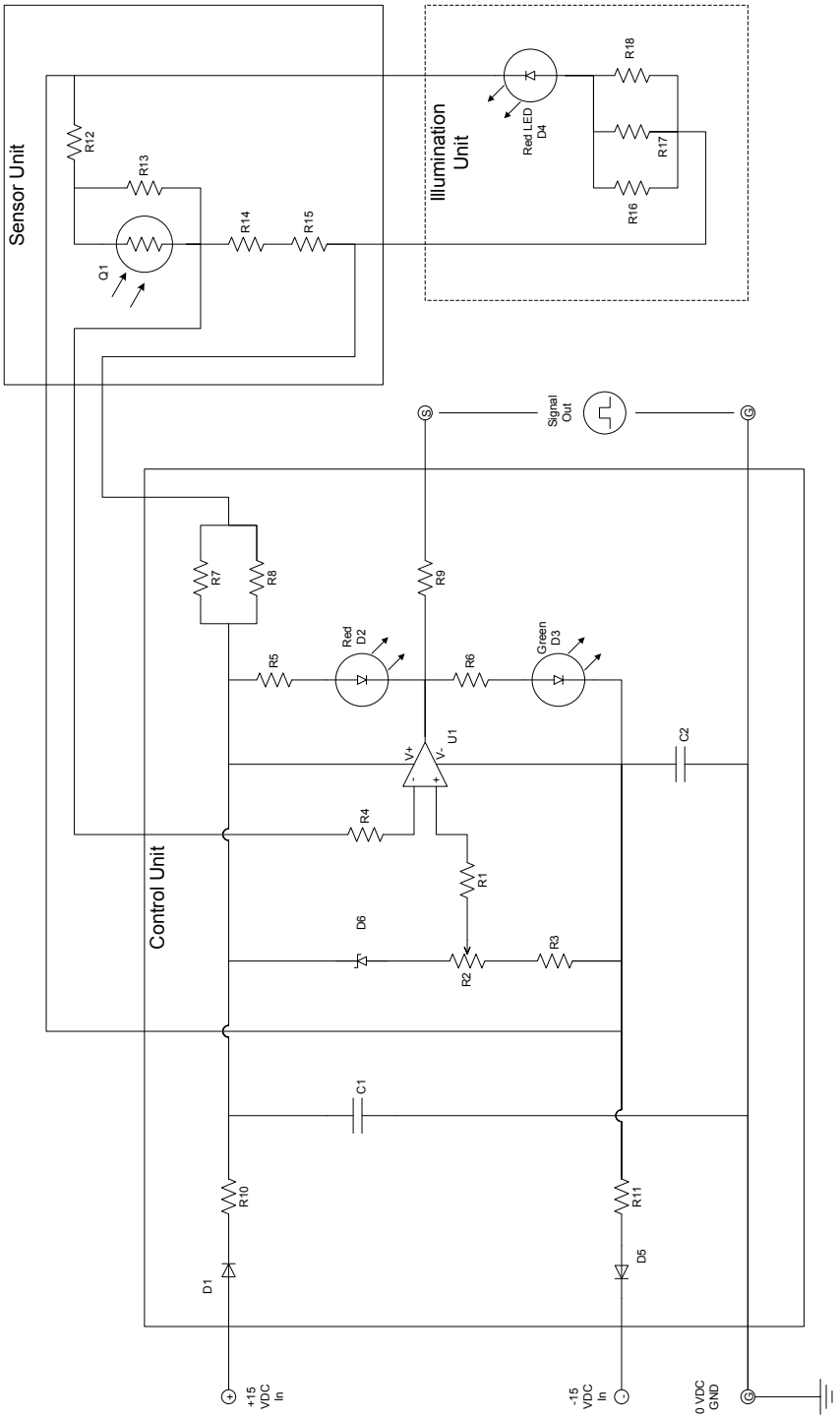

Figure 54: Cavitation Sensor Schematic

The electrical schematic for the cavitation sensor is given in Figure 54. Its corresponding components are listed in Table 16.

Table 16: Cavitation Sensor Components

C1: 560 pF	R3: 470 Ω
C2: 560 pF	R4: 1k Ω
	R5: 8.2k Ω
D1: 50SQ100	R6: 8.2k Ω
D2: HLMP-1700-B0002	R7: 33 Ω
D3: HLMP-1790-A0002	R8: 33 Ω
D4: Radioshack #276-0307 5mm 660nm 1.7V 20mA 3000mcd 12deg clear-lens LED	R9: 220 Ω
D5: 50SQ100	R10: 10 Ω
D6: 1N4615	R11: 10 Ω
	R12: 1.5k Ω
U1: MC34071A	R13: 47k Ω
	R14: 3.3k Ω
Q1: CdS Photocell – small one from Radioshack #276-1657 assortment (P1201?)	R15: 1.5k Ω
	R16: 10k Ω
R1: 1k Ω	R17: 10k Ω
R2: 150k Ω	R18: 10k Ω

A.5 SIGNAL INVERTER

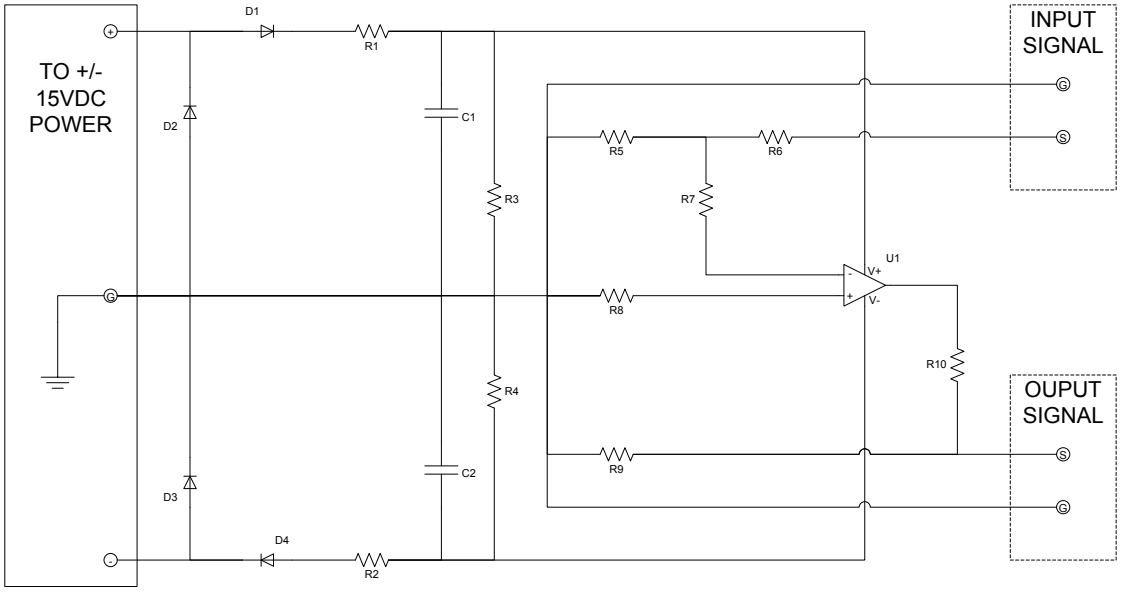

Figure 55: Signal Inverter Schematic

The electrical schematic for the signal inverter is given in Figure 55. The values for its components are given in Table 17.

Table 17: Signal Inverter Components

C1: 10 µF	R1: 1 Ω
C2: 10 µF	R2: 1 Ω
	R3: 15k Ω
D1: GP15M	R4: 15k Ω
D2: GP15M	R5: 10k Ω
D3: GP15M	R6: 4.7k Ω
D4: GP15M	R7: 10k Ω
	R8: 10k Ω
U1: MC34071AP	R9: 100k Ω
	R10: 1k Ω

A.6 SPEED CONTROLLER ELECTRONICS

An overview drawing of the subassemblies contained in the Speed Controller Electronics Unit is given in Figure 56. The Master Power subassembly is given in Figure 57, the Fault Detector subassembly is given in Figure 58, the Isolator subassembly is given in Figure 59, the Phase Detector subassembly is given in Figure 60, the Trigger subassembly is given in Figure 61, and the Waveform Adapter subassembly is given in Figure 62. The components used are listed in Table 18 and Table 19.

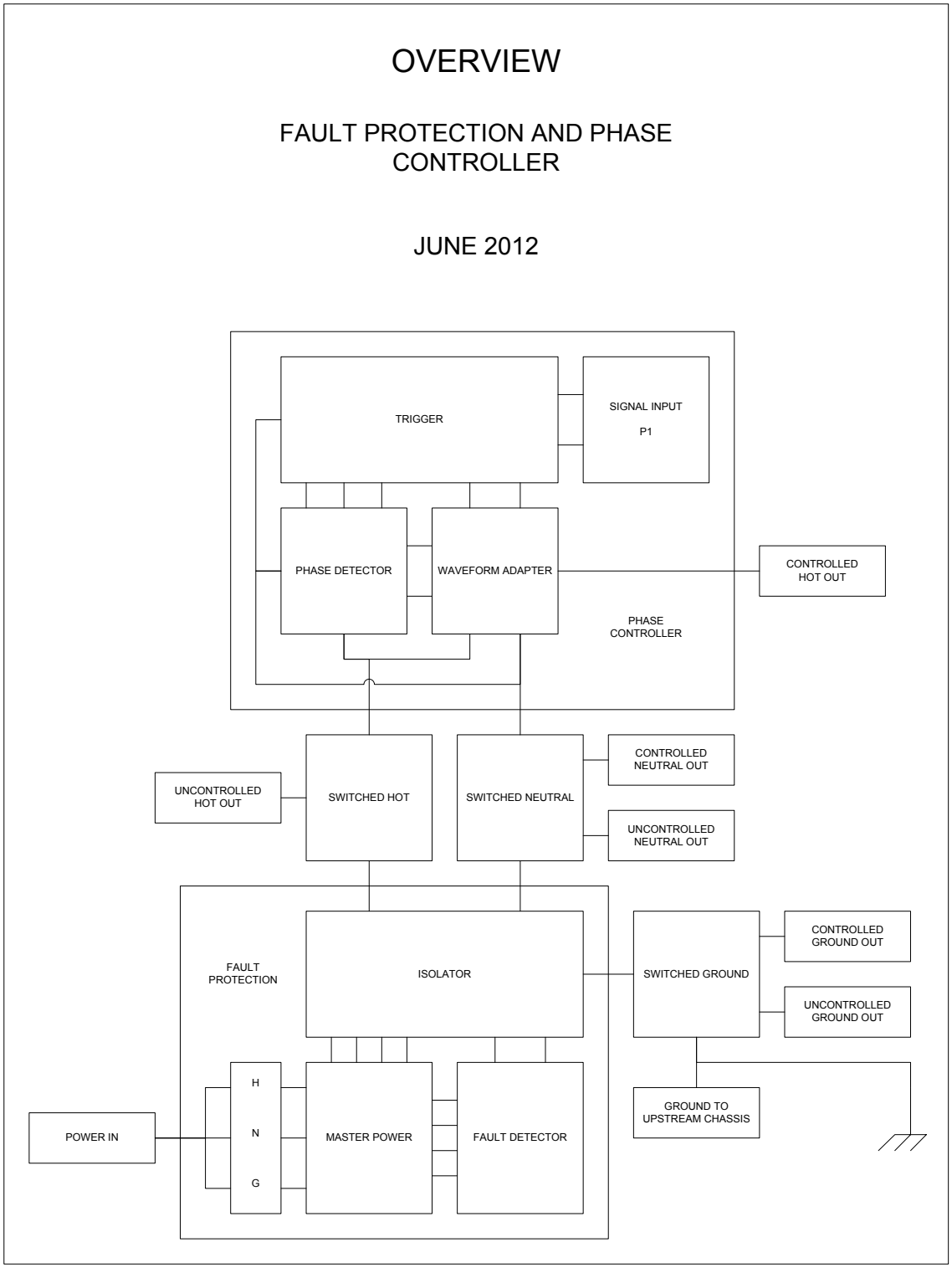

Figure 56: Speed Controller Electronics Overview

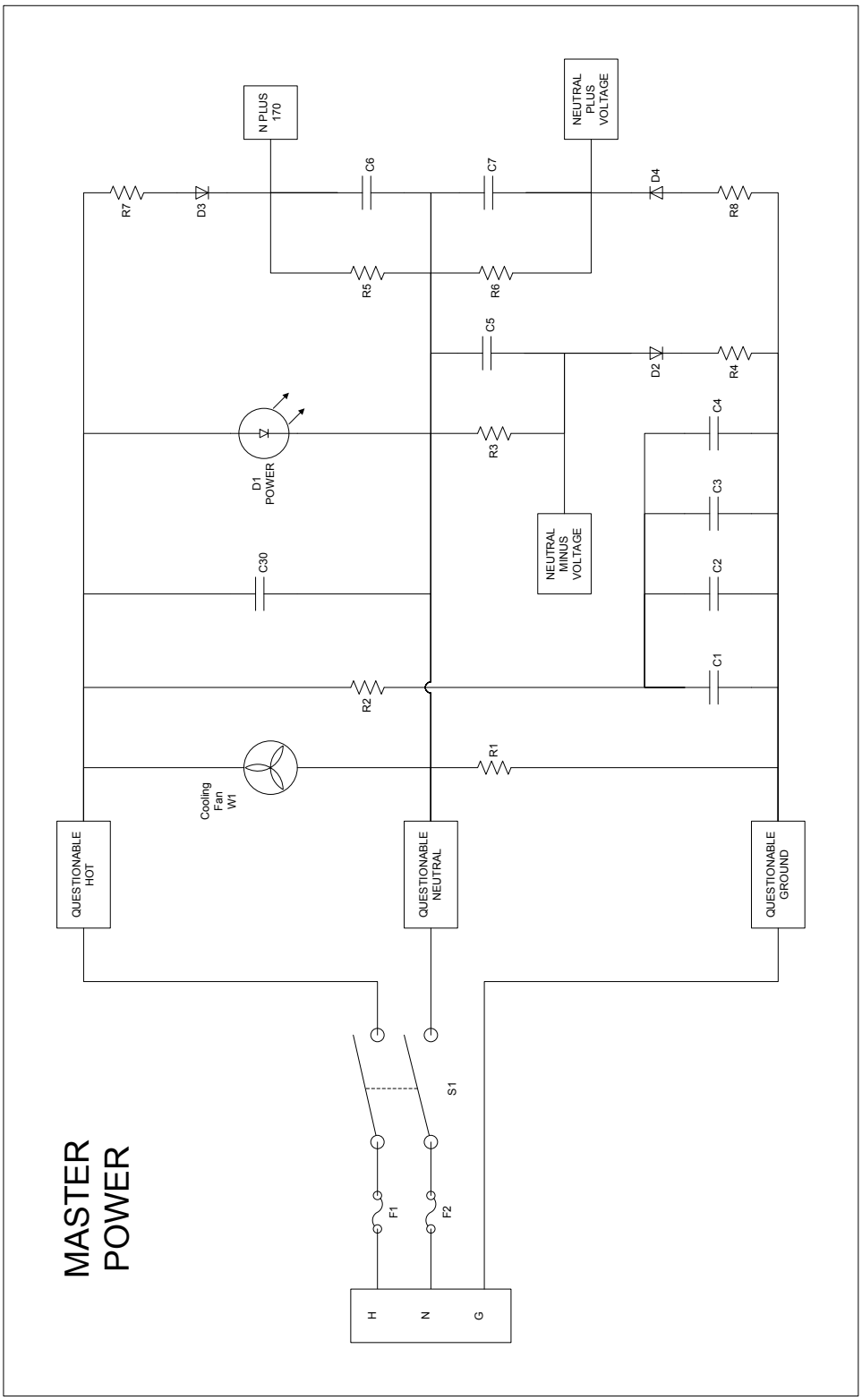

Figure 57: Master Power Subassembly

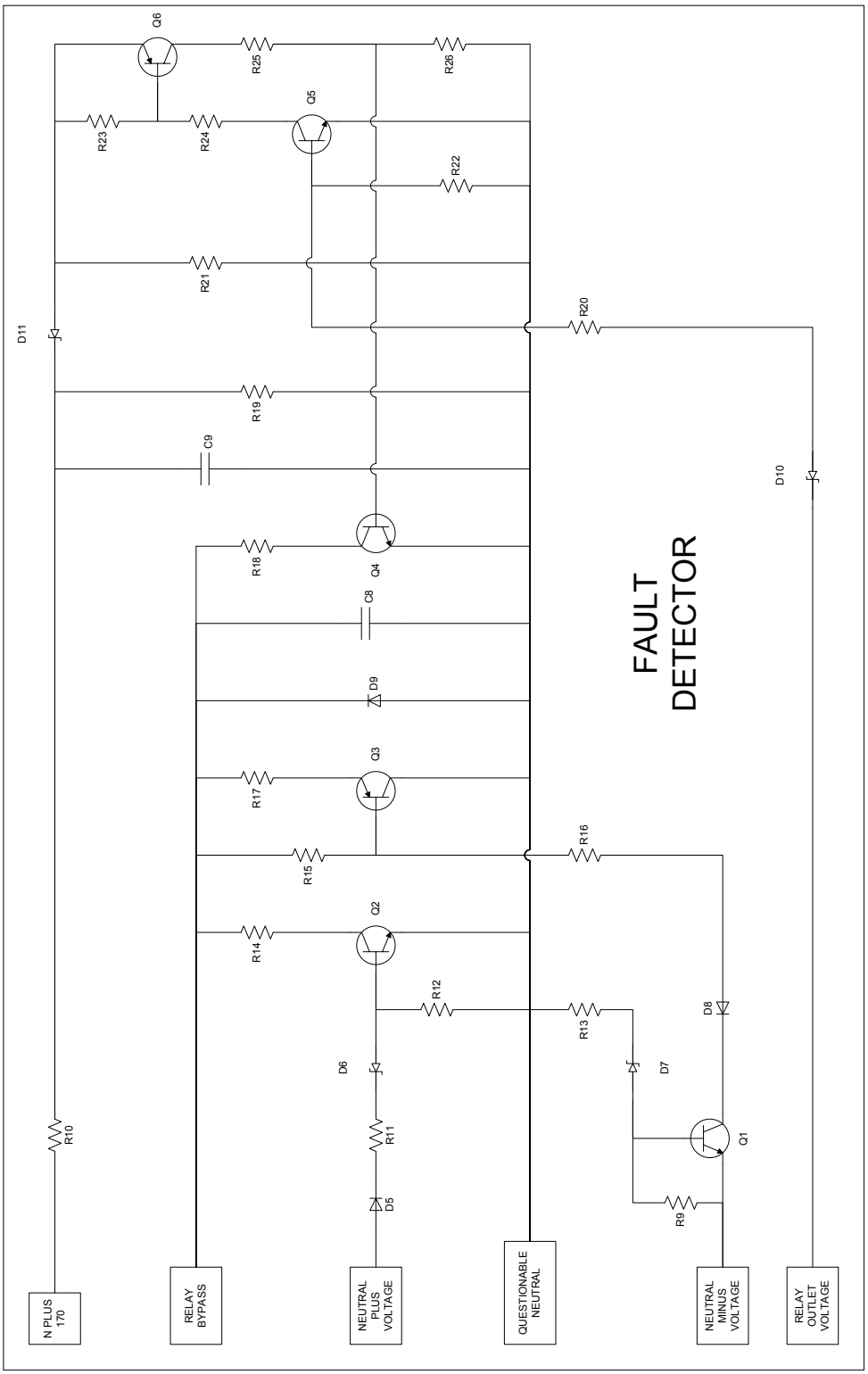

Figure 58: Fault Detector Subassembly

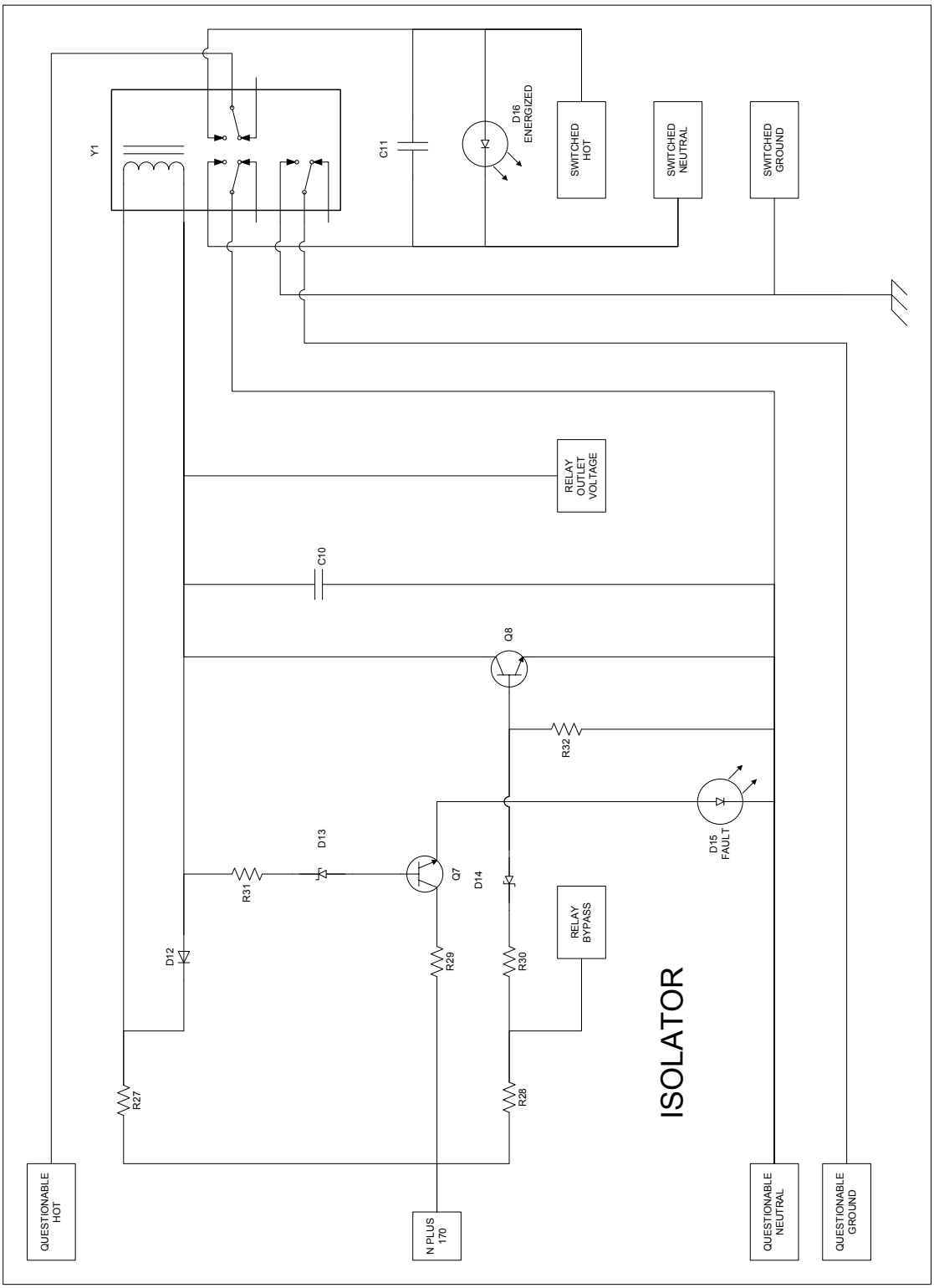

Figure 59: Isolator Subassembly

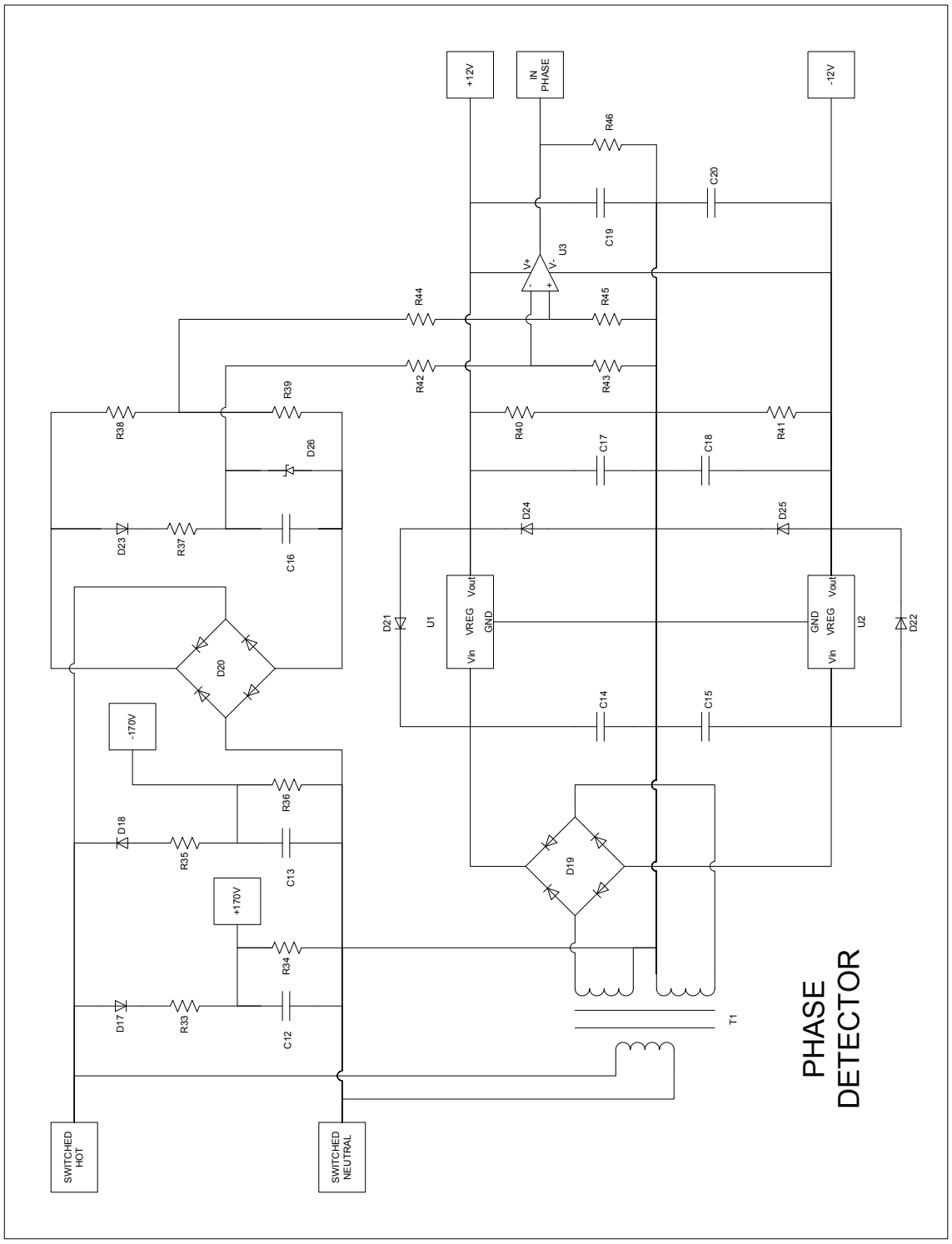

Figure 60: Phase Detector Subassembly

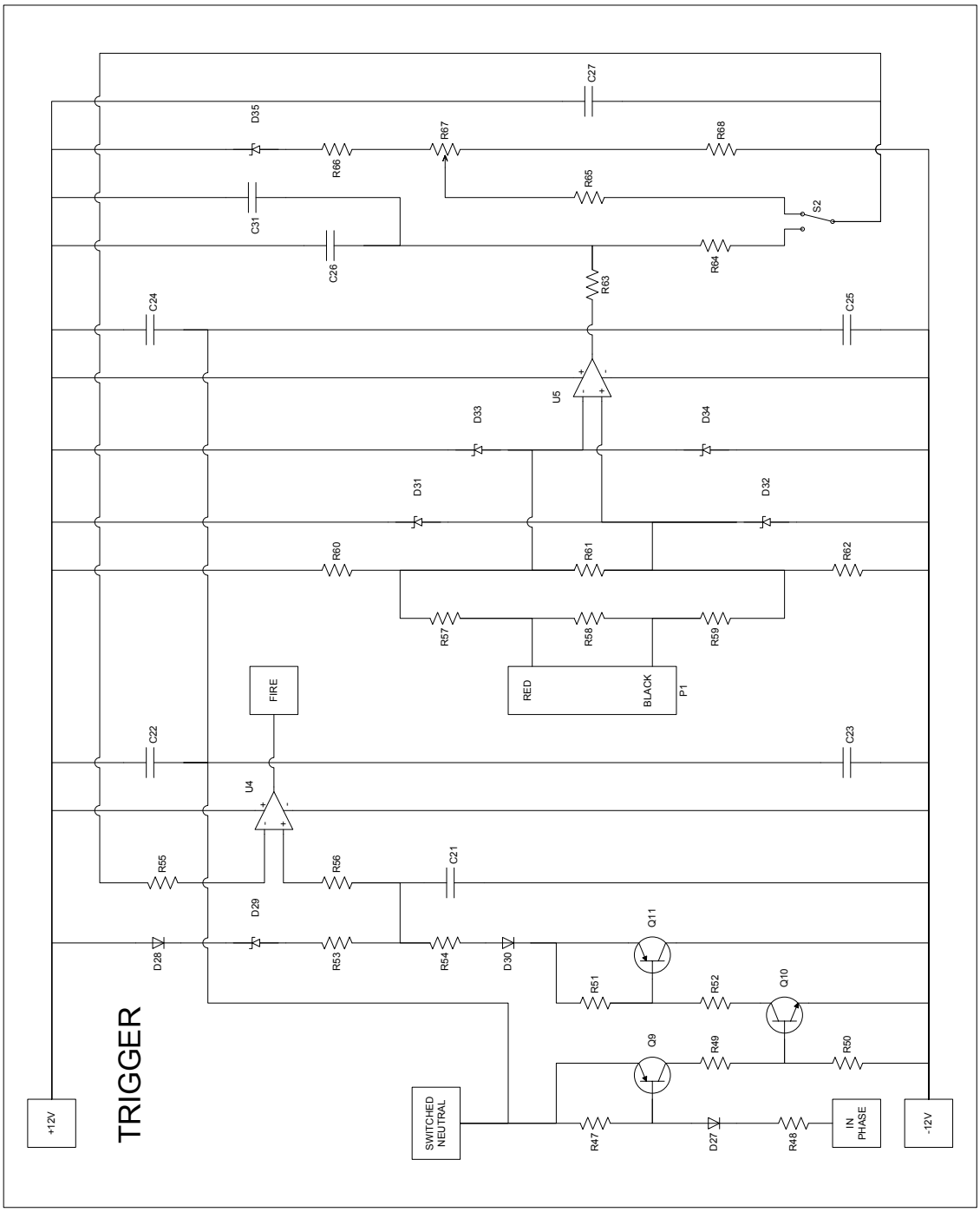

Figure 61: Trigger Subassembly

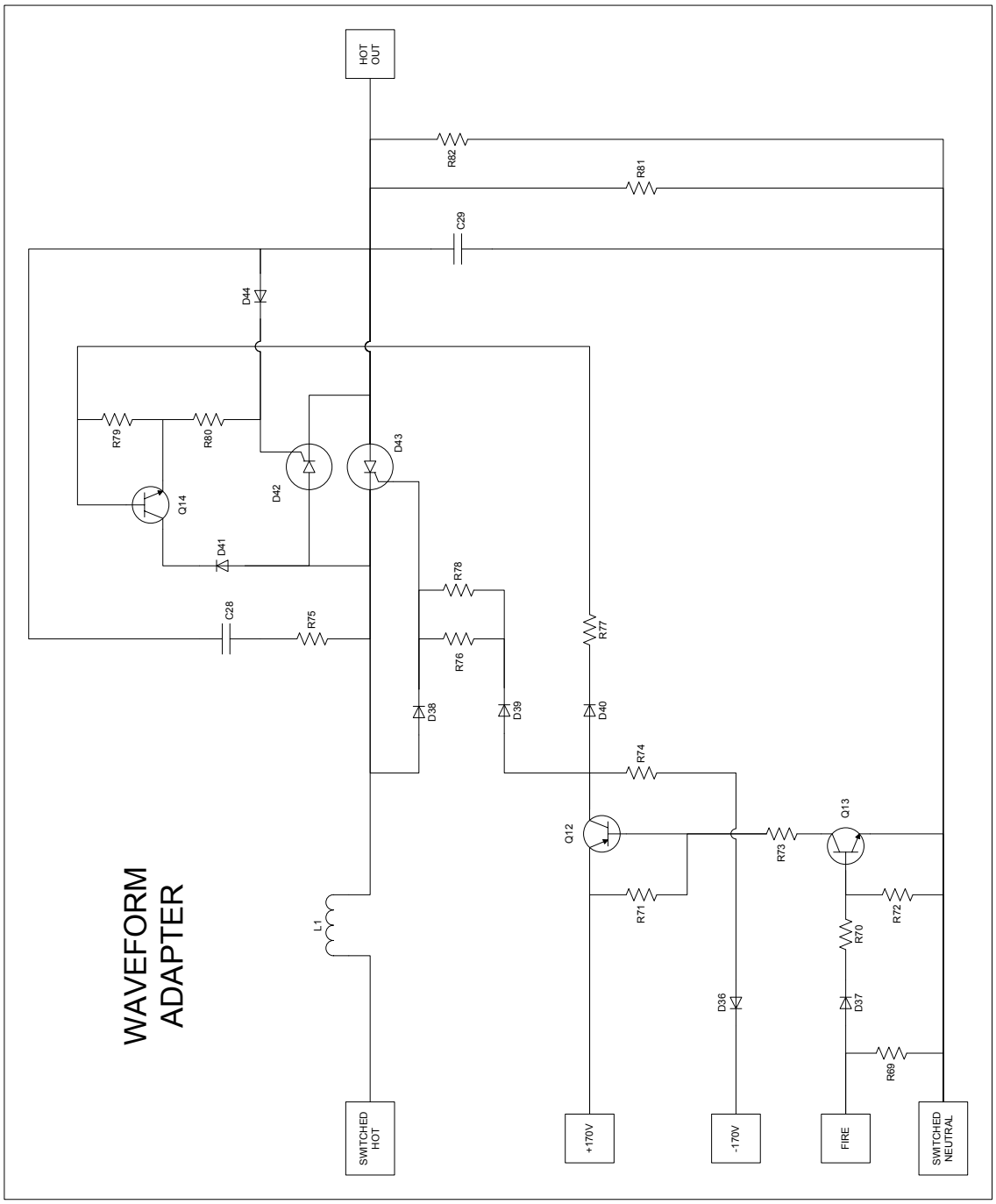

Figure 62: Waveform Adapter Subassembly

Table 18: Speed Controller Components, Part 1

C1: 0.01 µF	D2: 6A8-T	D35: 1N4615
C2: 0.01 µF	D3: 6A8-T	D36: 1N4005
C3: 0.01 µF	D4: 6A8-T	D37: STPS3150RL
C4: 0.01 µF	D5: 6A8-T	D38: 6A8-T
C5: 0.22 µF	D6: 1N4615	D39: 6A8-T
C6: 330 µF	D7: 1N4615	D40: 6A8-T
C7: 0.22 µF	D8: 6A8-T	D41: 6A8-T
C8: 47 µF	D9: 1N4004	D42: S6020L
C9: 33 µF	D10: 1N4742A	D43: S6020L
C10: 0.022 µF	D11: 1N4742A	D44: 6A8-T
C11: 0.1 µF	D12: 6A8-T	
C12: 330µF	D13: BZV85-CV2,113	F1: 10A SLO-BLO
C13: 22 µF	D14: BZV85-CV2,113	F2: 10A SLO-BLO
C14: 470 µF	D15: 5101H1	
C15: 470 µF	D16: 1091M5-125VAC	L1: 100 µH
C16: 4.7 µF	D17: 6A8-T	
C17: 47 µF	D18: 6A8-T	P1: DUAL BINDING POST
C18: 47 µF	D19: GBPC3506W-E4/51	
C19: 0.22 µF	D20: W10G	Q1: KSP45
C20: 0.22 µF	D21: STPS3150RL	Q2: 2N3417
C21: 0.1 µF	D22: STPS3150RL	Q3: BC557
C22: 0.22 µF	D23: 6A8-T	Q4: KSP45
C23: 0.22 µF	D24: STPS3150RL	Q5: KSP45
C24: 0.22 µF	D25: STPS3150RL	Q6: 2SA1381ESTU
C25: 0.22 µF	D26: 1N4615	Q7: KSP45
C26: 4.7 µF	D27: STPS3150RL	Q8: KSP45
C27: 47 nF	D28: STPS3150RL	Q9: KSH45H11ITU
C28: 0.1 µF	D29: 1N4615	Q10: 2N3417
C29: 47 nF	D30: STPS3150RL	Q11: KSH45H11ITU
C30: 0.1 µF	D31: BZX55B16	Q12: KSA1625K
C31: 10 µF	D32: BZX55B16	Q13: KSP45
	D33: BZX55B16	Q14: KSP45
D1: 1091M5-125VAC	D34: BZX55B16	

Table 19: Speed Controller Components, Part 2

R1: 10 MΩ	R34: 1 MΩ	R67: 150 kΩ 2W
R2: 100 Ω	R35: 10 Ω	R68: 2.2 kΩ
R3: 4.7 MΩ	R36: 1 MΩ	R69: 47 kΩ
R4: 33 kΩ	R37: 220 kΩ	R70: 3.3 kΩ
R5: 4.7 MΩ	R38: 15 kΩ 1%	R71: 330 kΩ
R6: 4.7 MΩ	R39: 200 kΩ 1%	R72: 100 kΩ
R7: 10 Ω	R40: 1 MΩ	R73: 47 kΩ 1W
R8: 33 kΩ	R41: 1 MΩ	R74: 1 MΩ
R9: 100 kΩ	R42: 10 MΩ 1%	R75: 15 Ω
R10: 300 kΩ	R43: 470 kΩ 1%	R76: 15 kΩ 5W
R11: 200 kΩ	R44: 10 MΩ 1%	R77: 100 kΩ 1W
R12: 100 kΩ	R45: 470 kΩ 1%	R78: 15 kΩ 5W
R13: 330 kΩ	R46: 15 kΩ	R79: 330 kΩ
R14: 47 Ω	R47: 4.7 kΩ	R80: 100 Ω
R15: 100 kΩ	R48: 15 kΩ	R81: 5.6 kΩ 5W
R16: 200 kΩ	R49: 1.5 kΩ	R82: 5.6 kΩ 5W
R17: 150 Ω	R50: 4.7 kΩ	
R18: 47 Ω	R51: 4.7 kΩ	S1: S821-RO
R19: 2.2 MΩ	R52: 220 Ω	S2: 103-R13-135C-02-EV
R20: 1 MΩ	R53: 27 kΩ 0.1%	
R21: 10 MΩ	R54: 9.1 Ω	T1: 164H36
R22: 2.2 MΩ	R55: 10 kΩ	
R23: 470 kΩ	R56: 10 kΩ	U1: LM7812
R24: 470 kΩ	R57: 15 kΩ	U2: LM7912
R25: 100 kΩ	R58: 2.2 kΩ	U3: MC34071APG
R26: 470 kΩ	R59: 15 kΩ	U4: MC34071APG
R27: 6.2 kΩ 5W	R60: 1 MΩ	U5: TLE2141ACP
R28: 75 kΩ 1W	R61: 2.2 kΩ	
R29: 16 kΩ 5W	R62: 1 MΩ	W1: OA80AP-11-2WB
R30: 1.6 kΩ	R63: 150 kΩ 0.1%	
R31: 470 kΩ	R64: 10 kΩ	Y1: KUP-14D15-110
R32: 100 kΩ	R65: 330 kΩ	
R33: 10 Ω	R66: 22 kΩ	

A.7 RS-232 ISOLATOR AND POWER SUPPLY

An overview of the RS-232 Isolator and Power supply is given in Figure 63. The subassemblies it describes are given in Figure 64, Figure 65, Figure 66, Figure 67, and Figure 68; the buffer subassemblies are identical, and only differ in their connections to the interface pins. The list of components is given in Table 20 and Table 21.

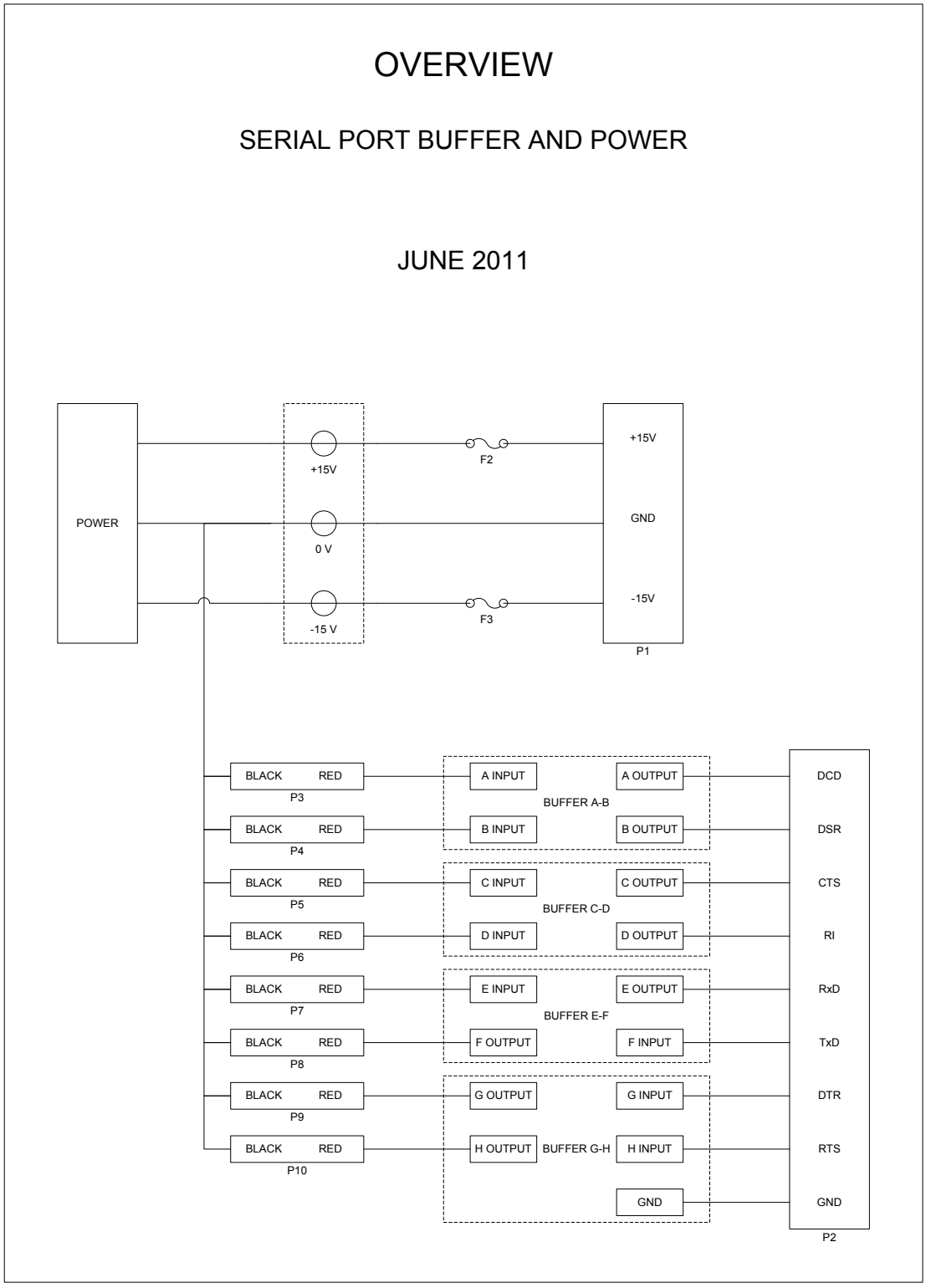

Figure 63: Serial Port Isolator Overview

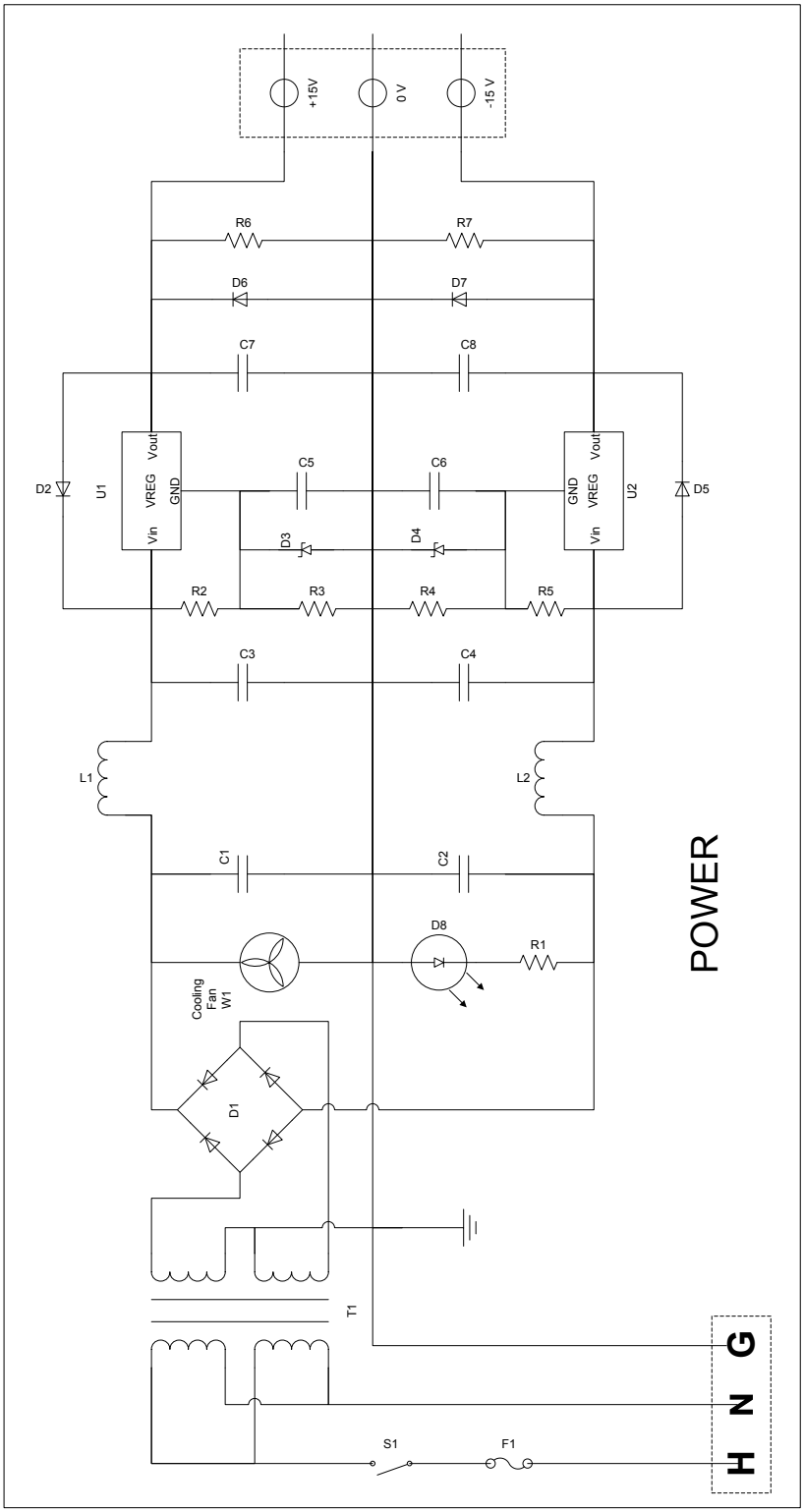

Figure 64: Power Supply Subassembly

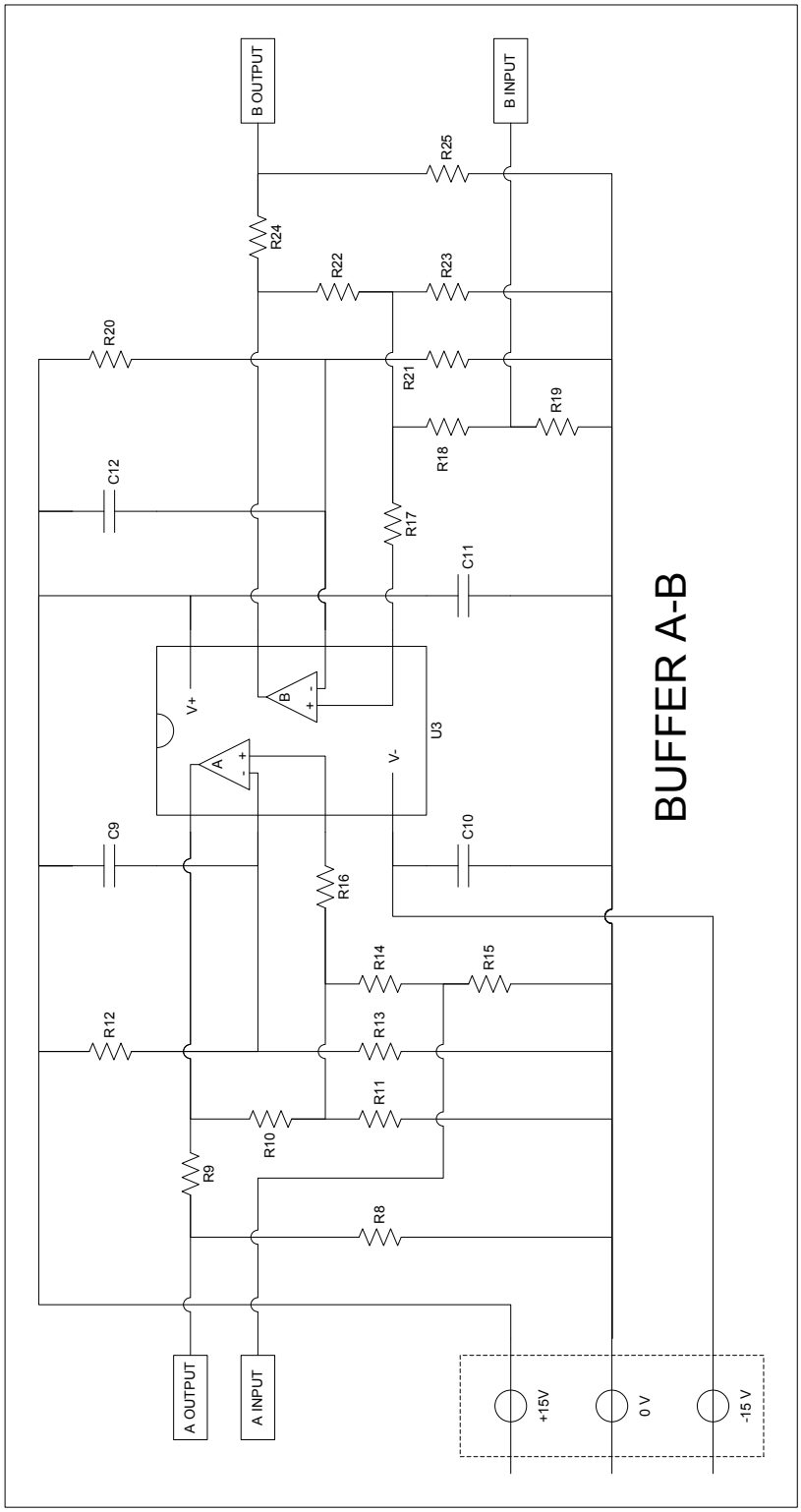

Figure 65: Buffer A-B Subassembly

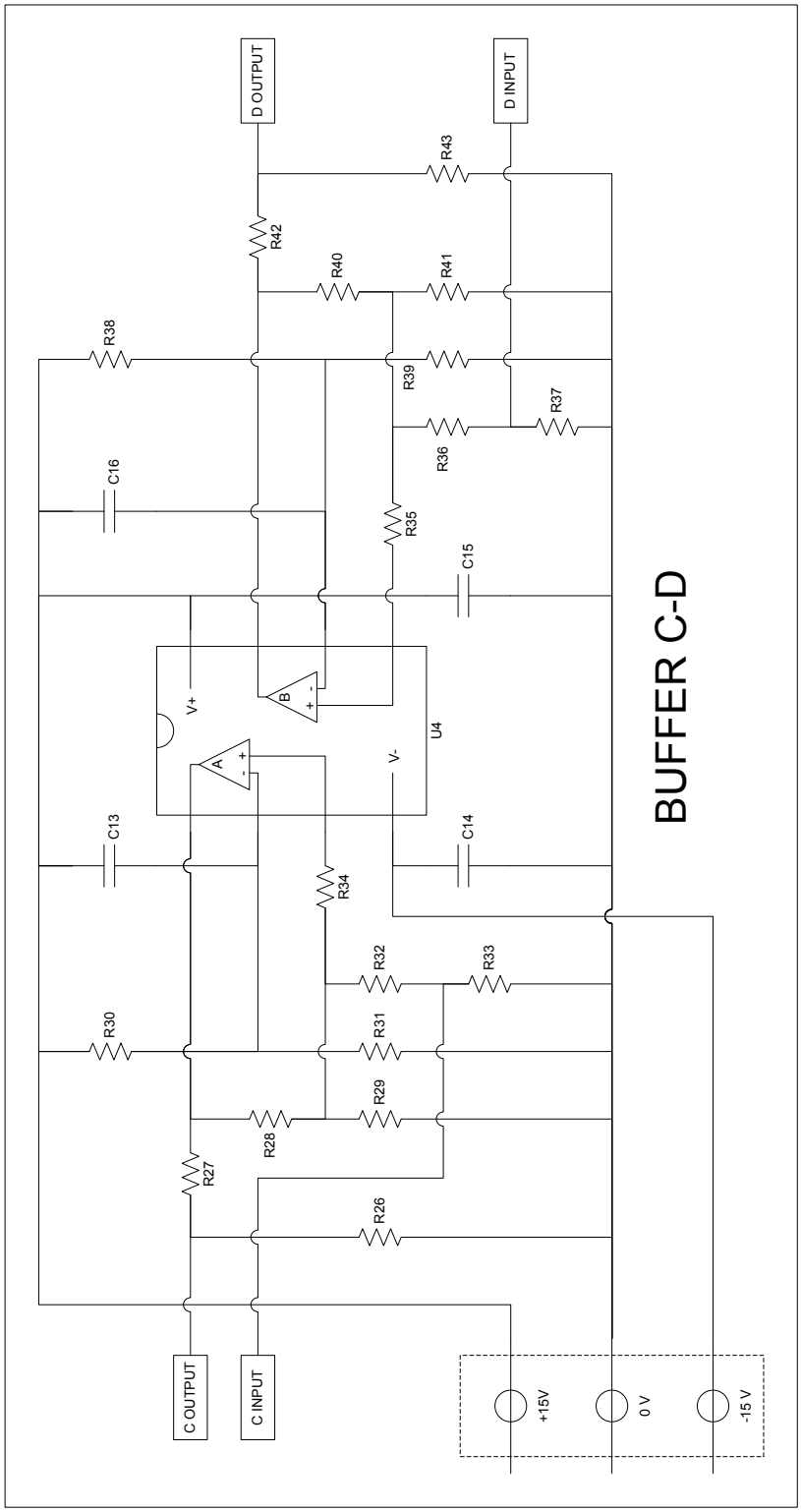

Figure 66: Buffer C-D Subassembly

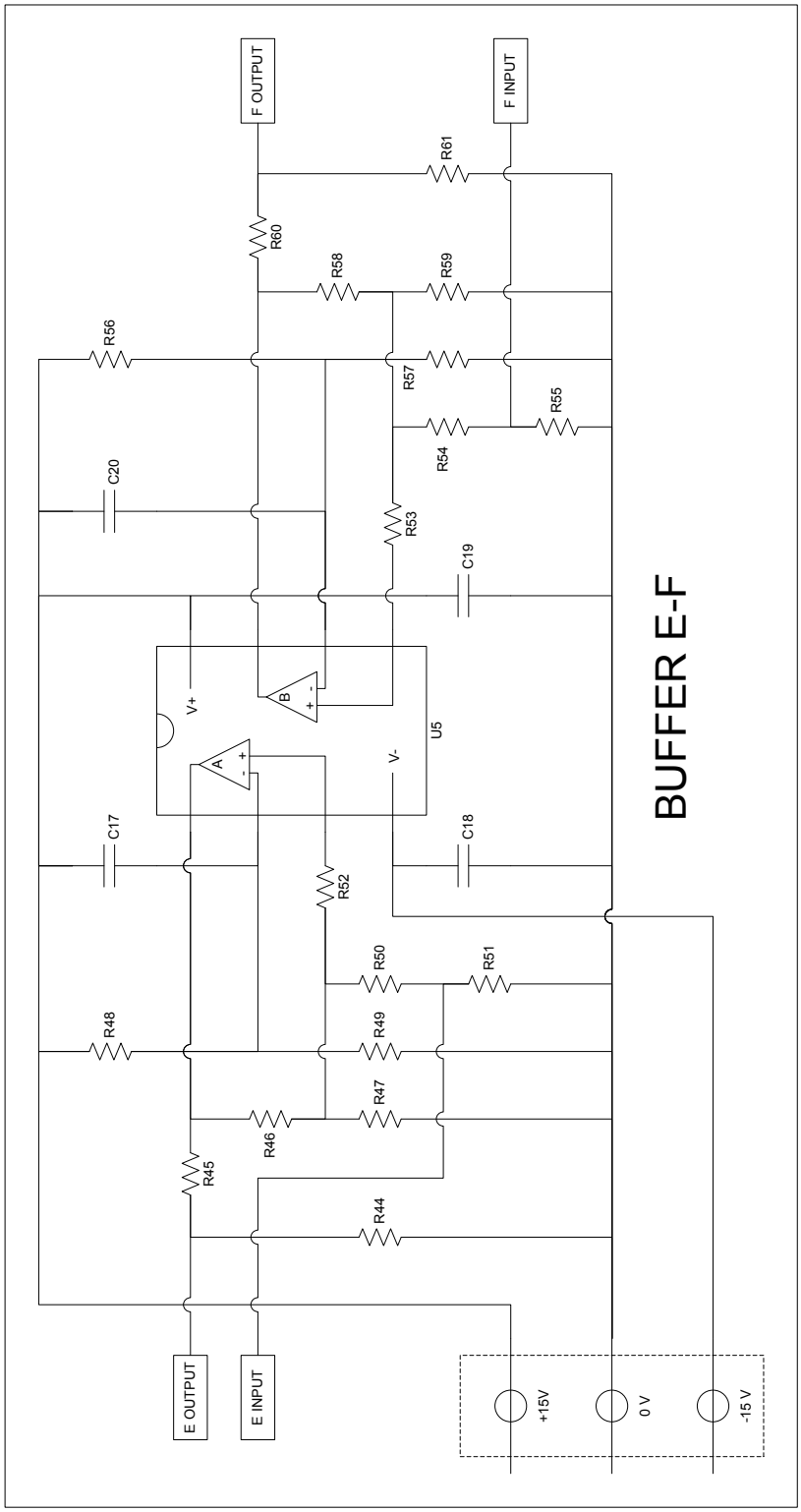

Figure 67: Buffer E-F Subassembly

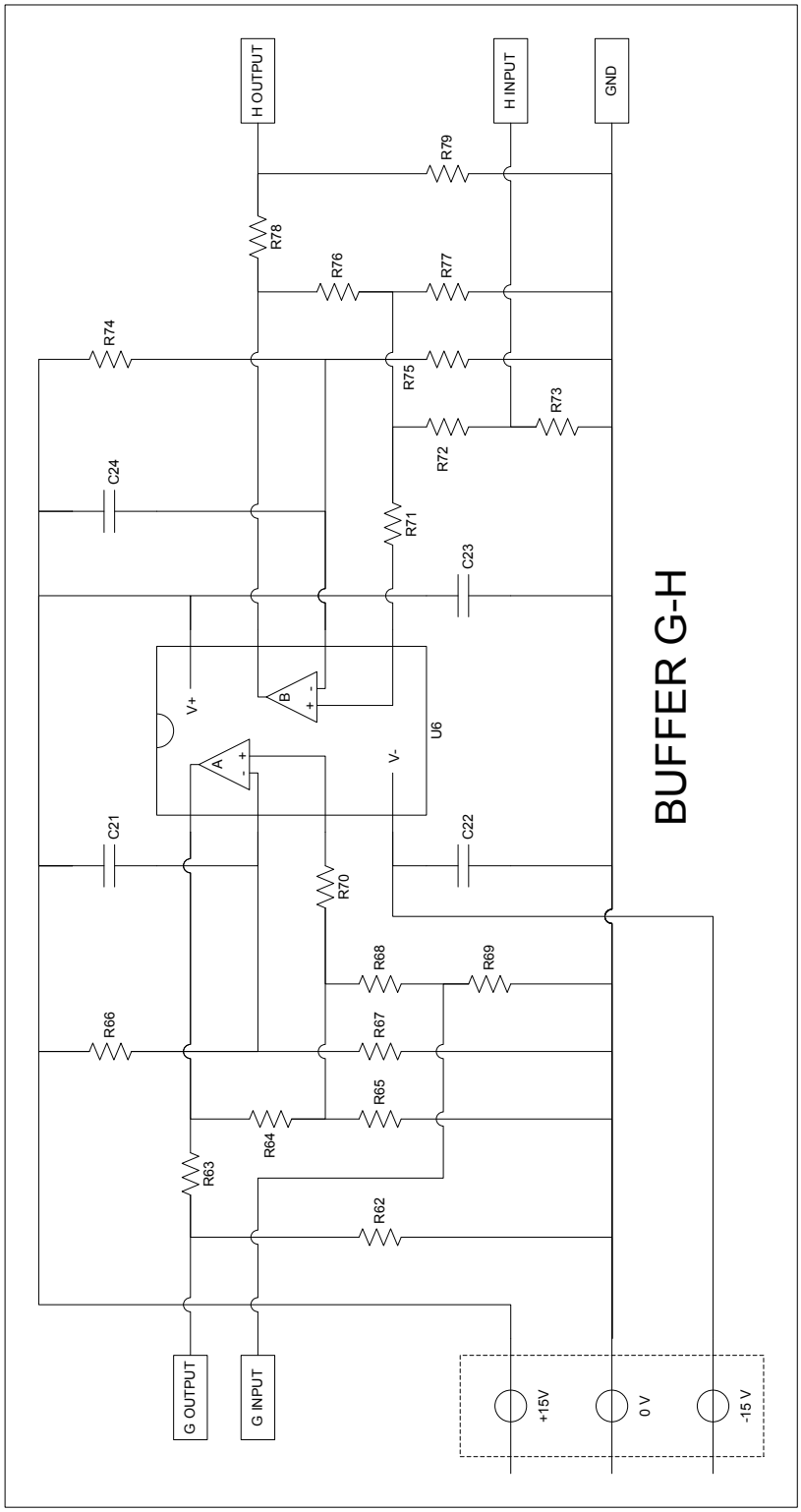

Figure 68: Buffer G-H Subassembly

Table 20: Serial Port Isolator Components, Part 1

C1: 3300 µF	D1: GBPC3506W-E4/51	P9: DUAL BINDING POST
C2: 3300 µF	D2: 50SQ100	P10: DUAL BINDING POST
C3: 3300 µF	D3: 1N5225B	
C4: 3300 µF	D4: 1N5225B	R1: 1.8K Ω 1W
C5: 470 µF	D5: 50SQ100	R2: 5.6K Ω
C6: 470 µF	D6: 50SQ100	R3: 2.2M Ω
C7: 68 µF	D7: 50SQ100	R4: 2.2M Ω
C8: 68 µF	D8: 5101H1	R5: 5.6K Ω
C9: 0.22 µF		R6: 1M Ω
C10: 0.22 µF	F1: 2.5 A	R7: 1M Ω
C11: 0.22 µF	F2: 1 A	R8: 33K Ω
C12: 0.22 µF	F3: 1 A	R9: 220 Ω
C13: 0.22 µF		R10: 2.4M Ω
C14: 0.22 µF	L1: 680 µH	R11: 30K Ω
C15: 0.22 µF	L2: 680 µH	R12: 330K Ω
C16: 0.22 µF		R13: 160 Ω
C17: 0.22 µF	P1: TRIPLE BINDING POST	R14: 470K Ω
C18: 0.22 µF	P2: DE-9-F	R15: 33K Ω
C19: 0.22 µF	P3: DUAL BINDING POST	R16: 1K Ω
C20: 0.22 µF	P4: DUAL BINDING POST	R17: 1K Ω
C21: 0.22 µF	P5: DUAL BINDING POST	R18: 470K Ω
C22: 0.22 µF	P6: DUAL BINDING POST	R19: 33K Ω
C23: 0.22 µF	P7: DUAL BINDING POST	R20: 330K Ω
C24: 0.22 µF	P8: DUAL BINDING POST	R21: 160 Ω

Table 21: Serial Port Isolator Components, Part 2

R22: 2.4M Ω	R46: 2.4M Ω	R70: 1K Ω
R23: 30K Ω	R47: 30K Ω	R71: 1K Ω
R24: 220 Ω	R48: 330K Ω	R72: 470K Ω
R25: 33K Ω	R49: 160 Ω	R73: 33K Ω
R26: 33K Ω	R50: 470K Ω	R74: 330K Ω
R27: 220 Ω	R51: 33K Ω	R75: 160 Ω
R28: 2.4M Ω	R52: 1K Ω	R76: 2.4M Ω
R29: 30K Ω	R53: 1K Ω	R77: 30K Ω
R30: 330K Ω	R54: 470K Ω	R78: 220 Ω
R31: 160 Ω	R55: 33K Ω	R79: 33K Ω
R32: 470K Ω	R56: 330K Ω	
R33: 33K Ω	R57: 160 Ω	S1: S301T-RO
R34: 1K Ω	R58: 2.4M Ω	
R35: 1K Ω	R59: 30K Ω	T1: VPT30-3330
R36: 470K Ω	R60: 220 Ω	
R37: 33K Ω	R61: 33K Ω	U1: LM7812
R38: 330K Ω	R62: 33K Ω	U2: LM7912
R39: 160 Ω	R63: 220 Ω	U3: OPA2132PAG4
R40: 2.4M Ω	R64: 2.4M Ω	U4: OPA2132PAG4
R41: 30K Ω	R65: 30K Ω	U5: OPA2132PAG4
R42: 220 Ω	R66: 330K Ω	U6: OPA2132PAG4
R43: 33K Ω	R67: 160 Ω	
R44: 33K Ω	R68: 470K Ω	W1: AD0624LB-A70GL-LF
R45: 220 Ω	R69: 33K Ω	

A.8 LASER INTERLOCK SYSTEM

An overview of the laser interlock system is given in Figure 69. The subassemblies are shown in Figure 70, Figure 71, Figure 72, Figure 73, and Figure 74. The components for the system are listed in Table 22.

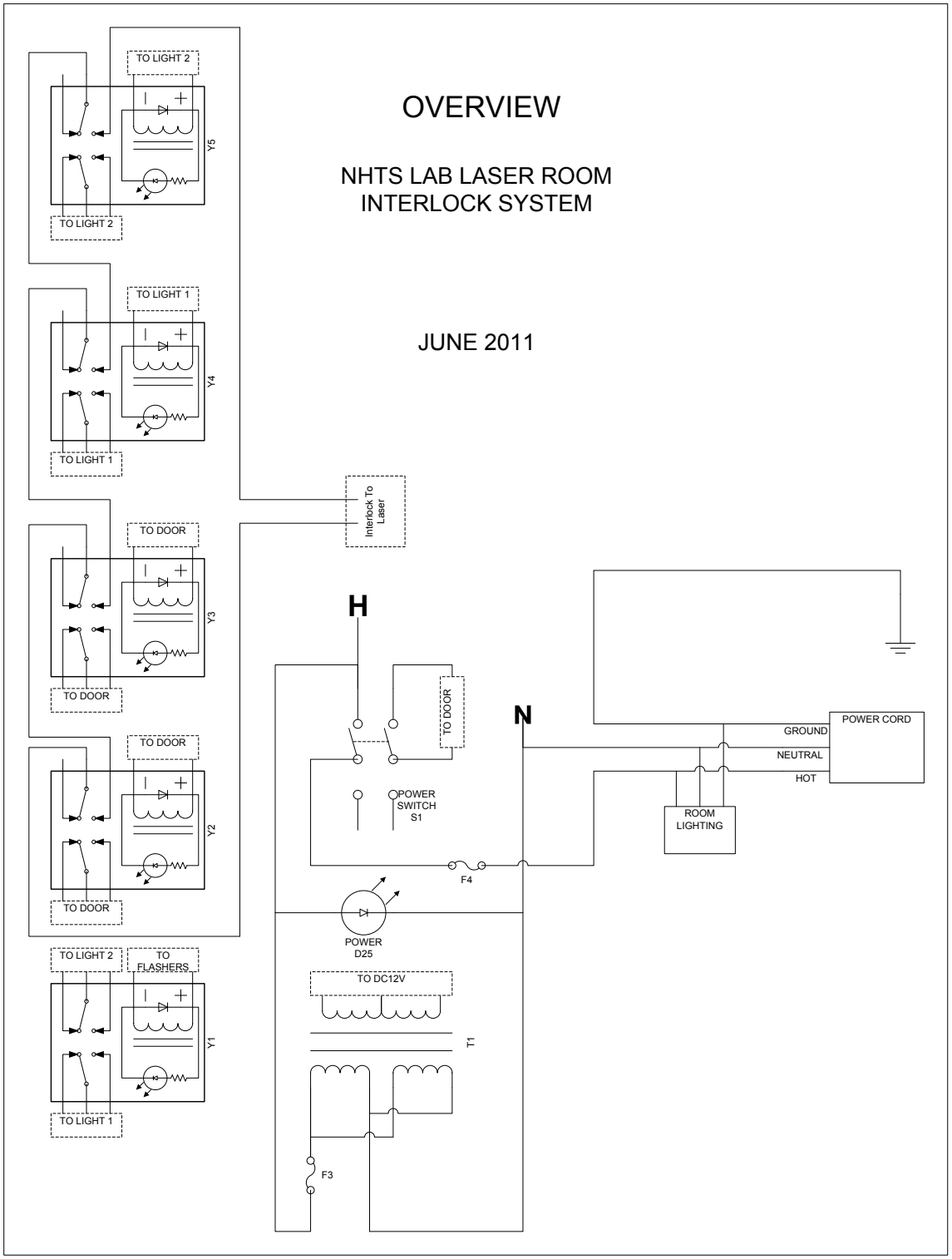

Figure 69: Laser Interlock System Overview

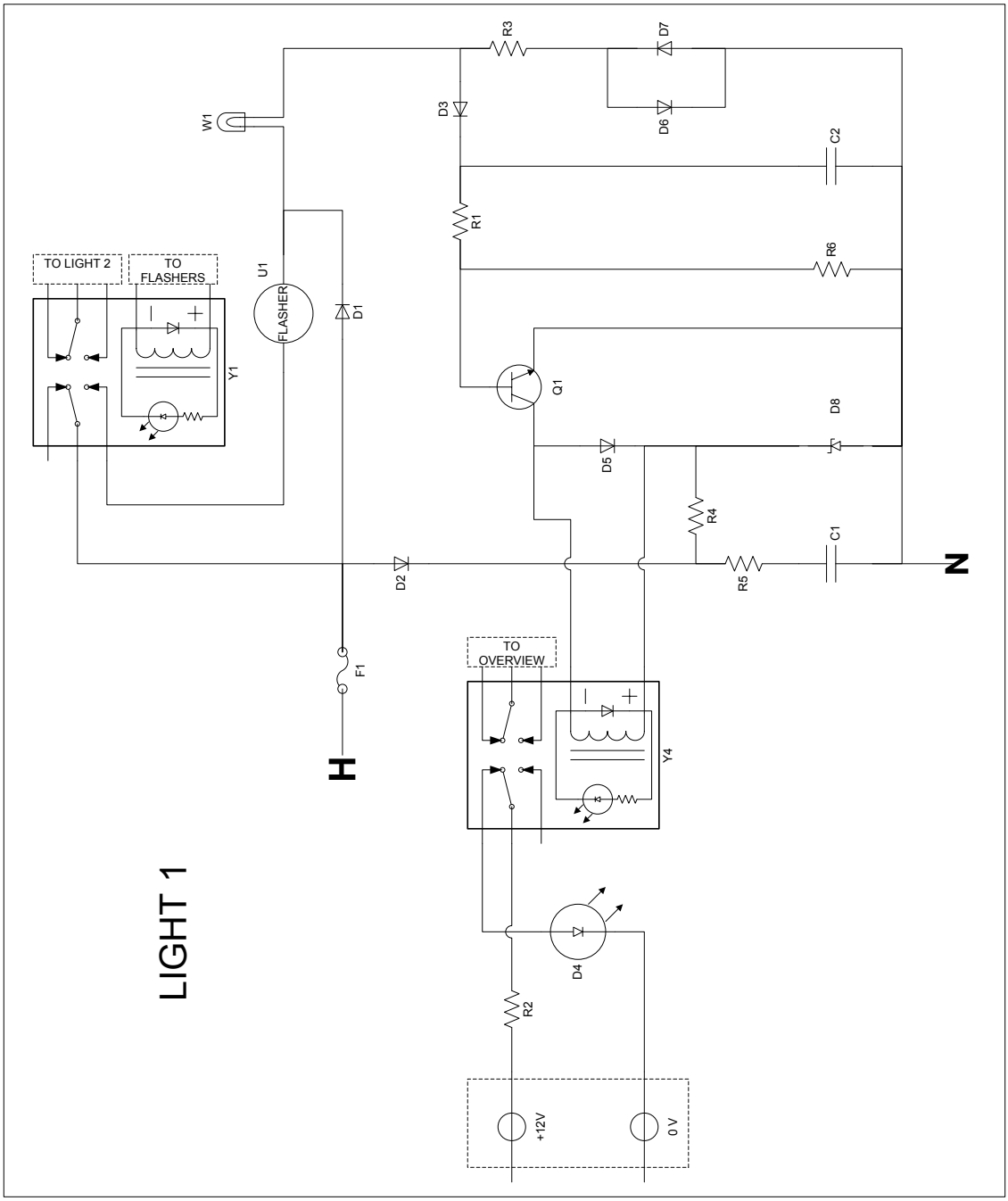

Figure 70: Interlock Light 1 Subassembly

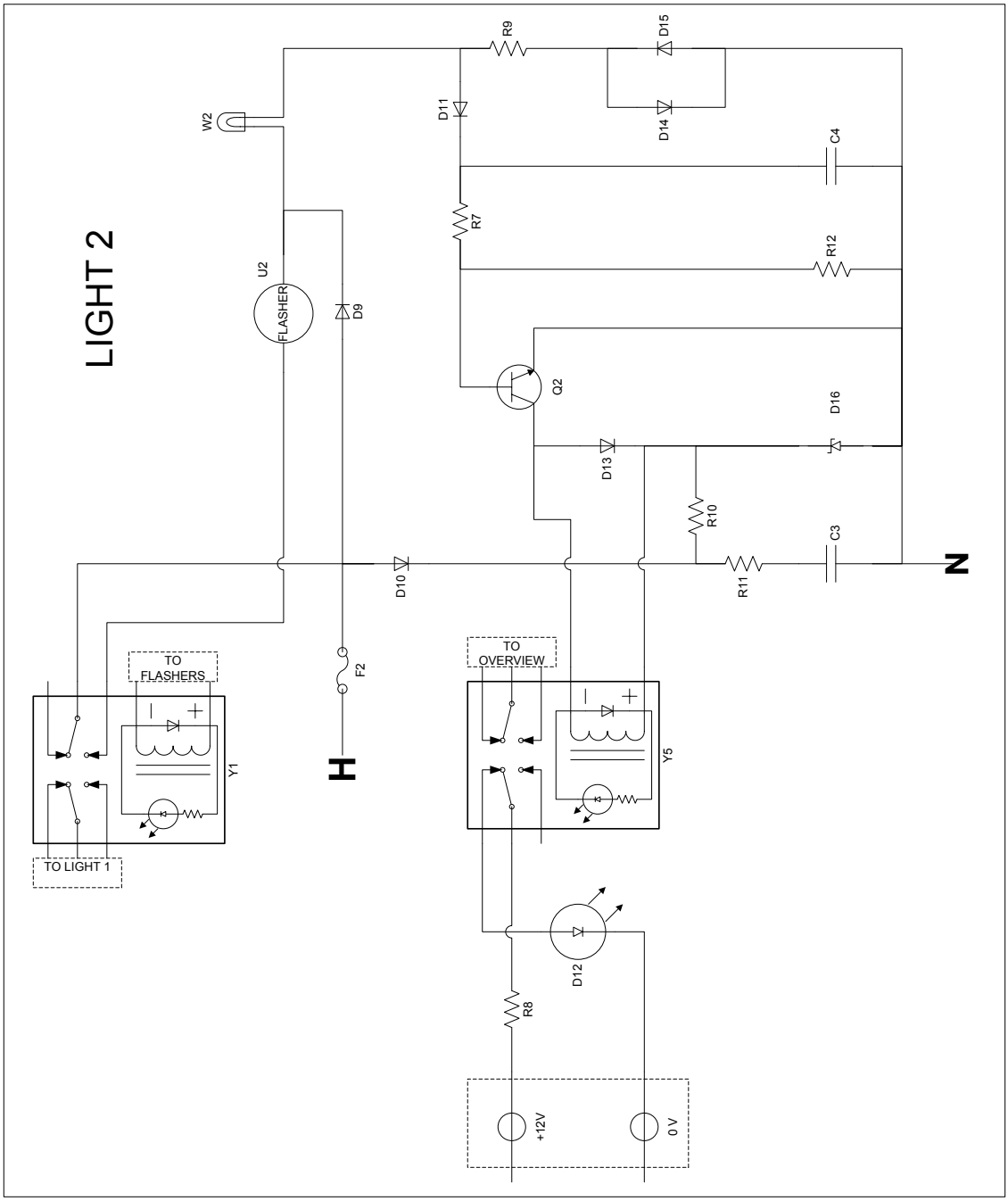

Figure 71: Interlock Light 2 Subassembly

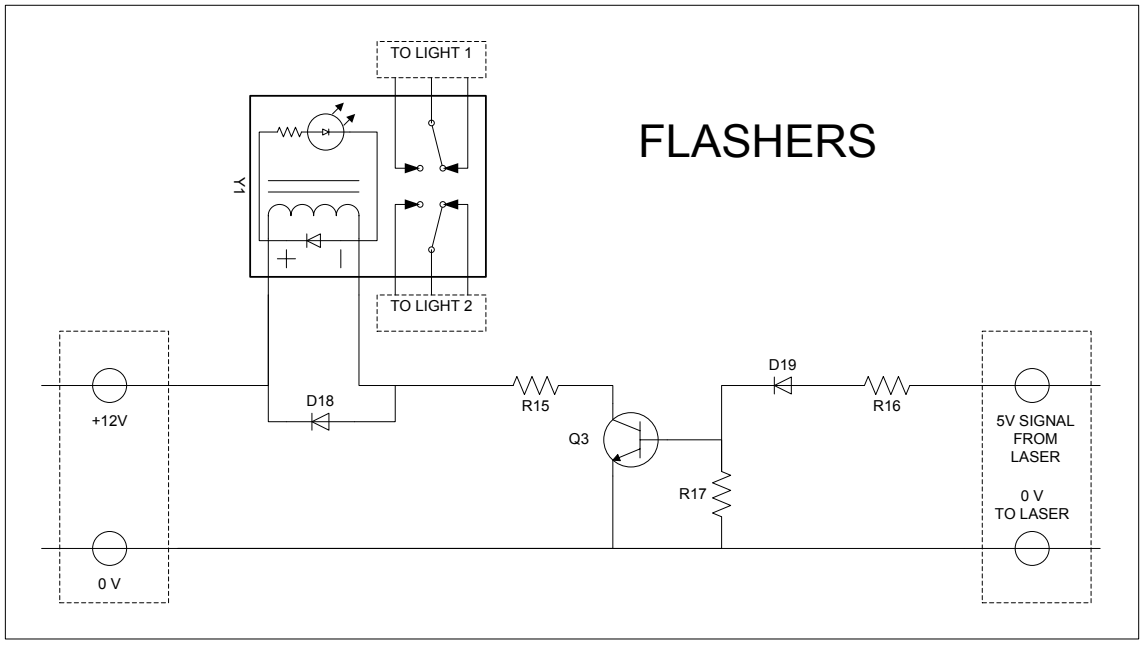

Figure 72: Interlock Light Flasher Controller Subassembly

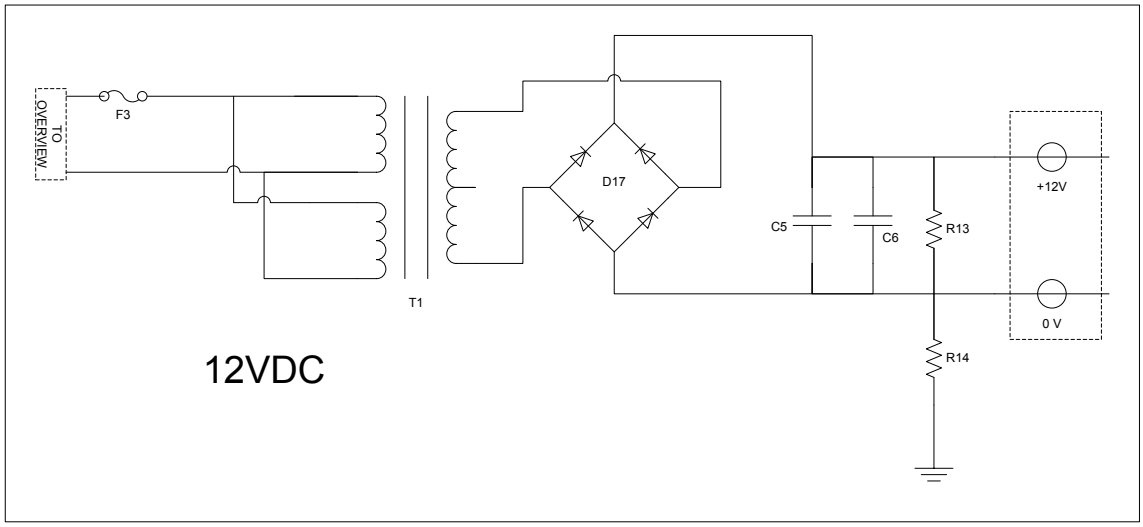

Figure 73: Interlock 12V Supply Subassembly

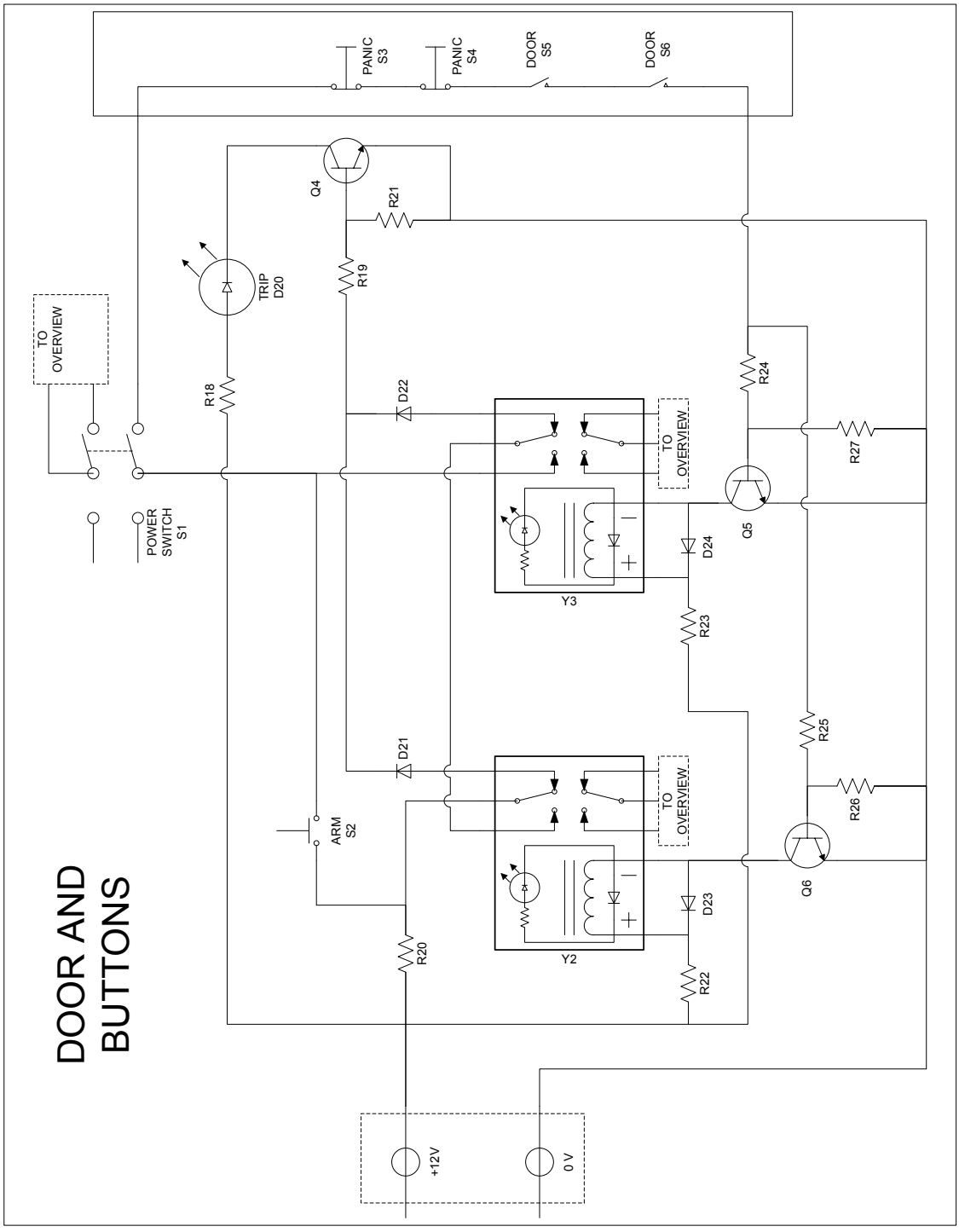

Figure 74: Interlock Door and Buttons Subassembly

Table 22: Laser Interlock System Components

C1: 220 µF	F1: 7A	R19: 750 Ω
C2: 1000 µF	F2: 7A	R20: 47 Ω 10 W
C3: 220 µF	F3: 1.25A	R21: 470K Ω
C4: 1000 µF	F4: 10A	R22: 18 Ω
		R23: 18 Ω
D1: 10A07	Q1: NTE128	R24: 750 Ω
D2: 10A07	Q2: NTE128	R25: 750 Ω
D3: 1N4004	Q3: NTE128	R26: 470K Ω
D4: 1091M7-12V	Q4: NTE128	R27: 470K Ω
D5: 1N4004	Q5: NTE128	
D6: 10A07	Q6: NTE128	S1: H2011U2F205NQ
D7: 10A07		S2: McMaster # 75759K21
D8: 1N5927BG	R1: 180 Ω	S3: McMaster # 6785K21
D9: 10A07	R2: 68 Ω	S4: McMaster # 6785K21
	R3: 4.7 Ω 10 W –	S5: Magnetic/Reed Switch
D10: 10A07	30J4R7E	505-101-WS
	R4: 2K Ω 50 W –	S6: Magnetic/Reed Switch
D11: 1N4004	RH0502K000FE02	505-101-WS
D12: 1091M7-12V	R5: 10 Ω	
D13: 1N4004	R6: 470K Ω	T1: 186D10
D14: 10A07	R7: 180 Ω	
D15: 10A07	R8: 68 Ω	U1: ETN-120-AFT-75
	R9: 4.7 Ω 10 W –	
D16: 1N5927BG	30J4R7E	U2: ETN-120-AFT-75
	R10: 2K Ω 50 W –	
D17: DFB2520	RH0502K000FE02	
		W1: 30-60W
D18: 1N4004	R11: 10 Ω	INCANDESCENT BULB
		W2: 30-60W
D19: 1N4004	R12: 470K Ω	INCANDESCENT BULB
D20: 1091M7-12V	R13: 470K Ω	
D21: 1N4004	R14: 470K Ω	Y1: HJ2-L-T-DC12V
D22: 1N4004	R15: 18 Ω	Y2: HJ2-L-T-DC12V
D23: 1N4004	R16: 390 Ω	Y3: HJ2-L-T-DC12V
D24: 1N4004	R17: 470K Ω	Y4: HJ2-L-T-DC12V
D25: 1091M1-125VAC	R18: 68 Ω	Y5: HJ2-L-T-DC12V

A.9 PULSE GENERATOR

The electrical schematic for the Pulse Generator is given in Figure 75. The components for it are listed in Table 23.

Table 23: Pulse Generator Components

C1: 10 µF	R1: 1 Ω
C2: 47 µF	R2: 1 MΩ
C3: 0.022 µF	R3: 1 MΩ
C4: 0.001 µF	R4: 1 kΩ
C5: 220 pF	R5: 68 Ω
C6: 100 pF	R6: 10 kΩ
C7: 10 µF	R7: 10 kΩ
C8: 100 pF	R8: 100 kΩ
C9: 0.001 µF	R9: 10 kΩ
	R10: 10 kΩ
D1: STPS5L40RL	R11: 27 kΩ
D2: STPS5L40RL	R12: 100 kΩ audio taper
D3: STPS5L40RL	R13: 1 kΩ
D4: 1N4614	R14: 1 kΩ
D5: BAT43	R15: 1 kΩ
D6: BAT43	R16: 39 Ω
D7: BAT43	R17: 100 kΩ
	R18: 1 kΩ
U1: LM7805	R19: 1.5 kΩ
U2: TLE2141AC	R20: 4.7 kΩ
U3: NE556	R21: 12 kΩ
	R22: 3.3 kΩ
	R23: 1 kΩ

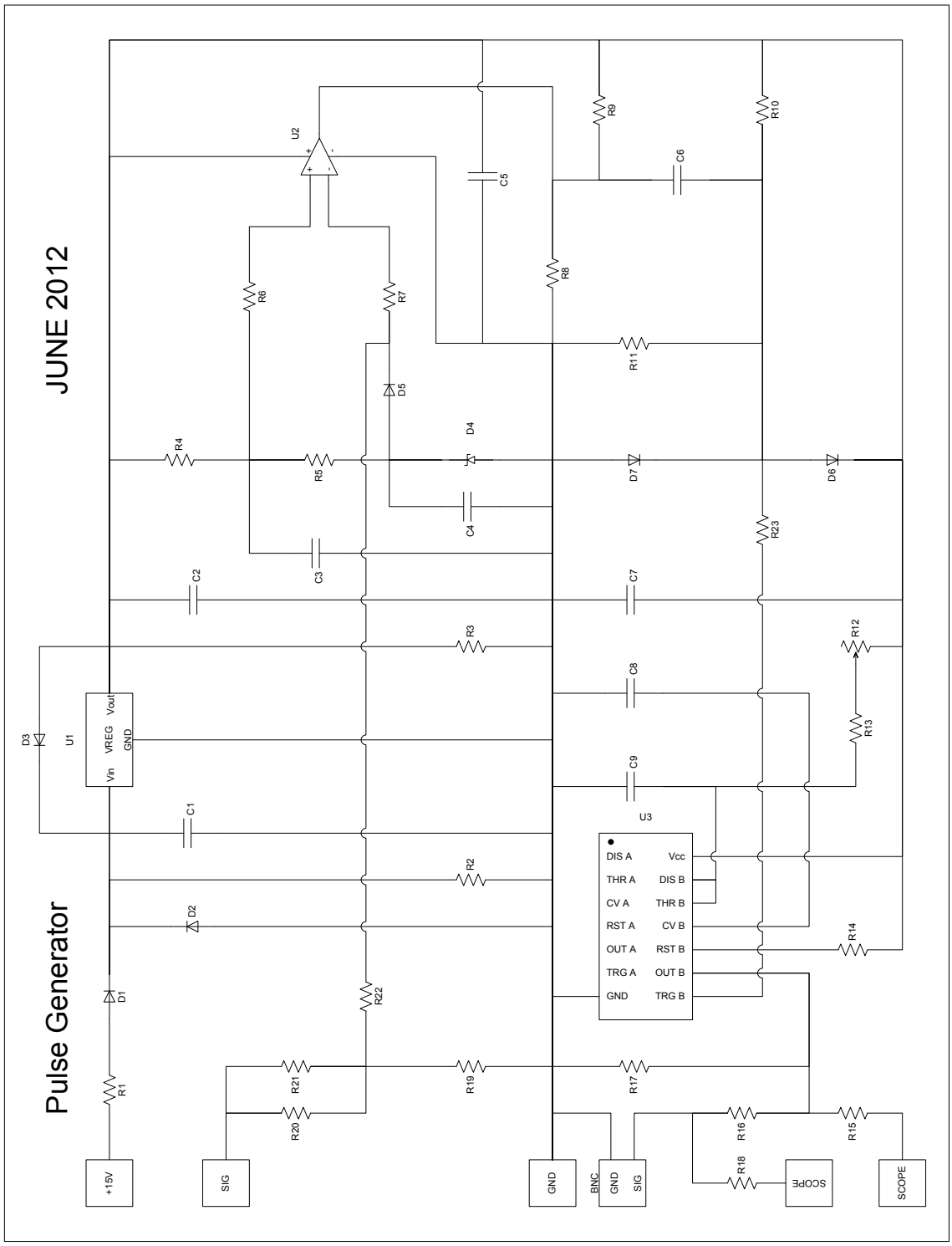

Figure 75: Pulse Generator Schematic

APPENDIX B

SPEEDCONTROL SOURCE CODE

A number of separate Visual Basic™ files were developed as part of the source code and User's Guide for the SpeedControl software. With the exception of the icons and certain resource files, they are all listed in this Appendix. A total of 21 files are included here.

B.1 ASSEMBLYINFO.VB

```
Imports System.Resources

Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("CTMFD Speed and Laser Control Application")>
<Assembly: AssemblyDescription("Control Application for the Speed and Laser in
the NHTS Lab CTMFD Facility")>
<Assembly: AssemblyCompany("Texas Engineering Experiment Station")>
<Assembly: AssemblyProduct("Speed and Laser Controller")>
<Assembly: AssemblyCopyright("Copyright © Texas A&M University 2011")>
<Assembly: AssemblyTrademark("Texas A&M University")>

<Assembly: ComVisible(False)>

'The following GUID is for the ID of the typelib if this project is exposed to
COM
<Assembly: Guid("083d8e3a-fda9-449a-a320-245cafc1a9e3")>

' Version information for an assembly consists of the following four values:
'
'      Major Version
'      Minor Version
'      Build Number
'      Revision
'
' You can specify all the values or you can default the Build and Revision
Numbers
' by using the '*' as shown below:
' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.*")>
<Assembly: AssemblyFileVersion("1.0.1.3")>

<Assembly: NeutralResourcesLanguageAttribute("en-US")>
```

B.2 FIREPULSE.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class FirePulse
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(FirePulse))
        Me.CommentBox = New System.Windows.Forms.TextBox
        Me.CancelPulseButton = New System.Windows.Forms.Button
        Me.FireButton = New System.Windows.Forms.Button
        Me.Label1 = New System.Windows.Forms.Label
        Me.Label2 = New System.Windows.Forms.Label
        Me.Label3 = New System.Windows.Forms.Label
        Me.SetValuesButton = New System.Windows.Forms.Button
        Me.DurationDisplay = New System.Windows.Forms.Label
        Me.PowerDisplay = New System.Windows.Forms.Label
        Me.DurationSetBox = New System.Windows.Forms.TextBox
        Me.PowerSetBox = New System.Windows.Forms.TextBox
        Me.SuspendLayout()
        '
        'CommentBox
        '
        Me.CommentBox.Location = New System.Drawing.Point(12, 25)
        Me.CommentBox.MaximumSize = New System.Drawing.Size(289, 74)
        Me.CommentBox.MinimumSize = New System.Drawing.Size(289, 74)
        Me.CommentBox.Multiline = True
        Me.CommentBox.Name = "CommentBox"
        Me.CommentBox.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.CommentBox.Size = New System.Drawing.Size(289, 74)
        Me.CommentBox.TabIndex = 5
        Me.CommentBox.Text = "(Leave a comment)"
        '
        'CancelPulseButton
        '
        Me.CancelPulseButton.Location = New System.Drawing.Point(226, 194)
        Me.CancelPulseButton.MaximumSize = New System.Drawing.Size(75, 23)
        Me.CancelPulseButton.MinimumSize = New System.Drawing.Size(75, 23)
        Me.CancelPulseButton.Name = "CancelPulseButton"
```

```
Me.CancelPulseButton.Size = New System.Drawing.Size(75, 23)
Me.CancelPulseButton.TabIndex = 4
Me.CancelPulseButton.Text = "Cancel"
Me.CancelPulseButton.UseVisualStyleBackColor = True
'
'FireButton
'
Me.FireButton.Location = New System.Drawing.Point(12, 194)
Me.FireButton.MaximumSize = New System.Drawing.Size(75, 23)
Me.FireButton.MinimumSize = New System.Drawing.Size(75, 23)
Me.FireButton.Name = "FireButton"
Me.FireButton.Size = New System.Drawing.Size(75, 23)
Me.FireButton.TabIndex = 3
Me.FireButton.Text = "FIRE!"
Me.FireButton.UseVisualStyleBackColor = True
'
'Label1
'
Me.Label1.AutoSize = True
Me.Label1.Location = New System.Drawing.Point(9, 9)
Me.Label1.MaximumSize = New System.Drawing.Size(54, 13)
Me.Label1.MinimumSize = New System.Drawing.Size(54, 13)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(54, 13)
Me.Label1.TabIndex = 6
Me.Label1.Text = "Comment:"
'
'Label2
'
Me.Label2.AutoSize = True
Me.Label2.Location = New System.Drawing.Point(18, 121)
Me.Label2.MaximumSize = New System.Drawing.Size(71, 13)
Me.Label2.MinimumSize = New System.Drawing.Size(71, 13)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(71, 13)
Me.Label2.TabIndex = 7
Me.Label2.Text = "Est. Duration:"
'
'Label3
'
Me.Label3.AutoSize = True
Me.Label3.Location = New System.Drawing.Point(49, 149)
Me.Label3.MaximumSize = New System.Drawing.Size(40, 13)
Me.Label3.MinimumSize = New System.Drawing.Size(40, 13)
Me.Label3.Name = "Label3"
Me.Label3.Size = New System.Drawing.Size(40, 13)
Me.Label3.TabIndex = 8
Me.Label3.Text = "Power:"
'
'SetValuesButton
'
Me.SetValuesButton.Location = New System.Drawing.Point(120, 194)
Me.SetValuesButton.MaximumSize = New System.Drawing.Size(75, 23)
Me.SetValuesButton.MinimumSize = New System.Drawing.Size(75, 23)
Me.SetValuesButton.Name = "SetValuesButton"
Me.SetValuesButton.Size = New System.Drawing.Size(75, 23)
Me.SetValuesButton.TabIndex = 9
Me.SetValuesButton.Text = "Set Values"
Me.SetValuesButton.UseVisualStyleBackColor = True
```

```
        '
        'DurationDisplay
        '
        Me.DurationDisplay.AutoSize = True
        Me.DurationDisplay.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.DurationDisplay.Location = New System.Drawing.Point(95, 121)
        Me.DurationDisplay.MaximumSize = New System.Drawing.Size(100, 15)
        Me.DurationDisplay.MinimumSize = New System.Drawing.Size(100, 15)
        Me.DurationDisplay.Name = "DurationDisplay"
        Me.DurationDisplay.Size = New System.Drawing.Size(100, 15)
        Me.DurationDisplay.TabIndex = 10
        Me.DurationDisplay.Text = "0.0001"
        '
        'PowerDisplay
        '
        Me.PowerDisplay.AutoSize = True
        Me.PowerDisplay.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.PowerDisplay.Location = New System.Drawing.Point(95, 149)
        Me.PowerDisplay.MaximumSize = New System.Drawing.Size(100, 15)
        Me.PowerDisplay.MinimumSize = New System.Drawing.Size(100, 15)
        Me.PowerDisplay.Name = "PowerDisplay"
        Me.PowerDisplay.Size = New System.Drawing.Size(100, 15)
        Me.PowerDisplay.TabIndex = 11
        Me.PowerDisplay.Text = "100"
        '
        'DurationSetBox
        '
        Me.DurationSetBox.Location = New System.Drawing.Point(201, 118)
        Me.DurationSetBox.MaximumSize = New System.Drawing.Size(100, 20)
        Me.DurationSetBox.MaxLength = 12
        Me.DurationSetBox.MinimumSize = New System.Drawing.Size(100, 20)
        Me.DurationSetBox.Name = "DurationSetBox"
        Me.DurationSetBox.Size = New System.Drawing.Size(100, 20)
        Me.DurationSetBox.TabIndex = 12
        Me.DurationSetBox.Text = "0.0001"
        '
        'PowerSetBox
        '
        Me.PowerSetBox.Location = New System.Drawing.Point(201, 146)
        Me.PowerSetBox.MaximumSize = New System.Drawing.Size(100, 20)
        Me.PowerSetBox.MaxLength = 9
        Me.PowerSetBox.MinimumSize = New System.Drawing.Size(100, 20)
        Me.PowerSetBox.Name = "PowerSetBox"
        Me.PowerSetBox.Size = New System.Drawing.Size(100, 20)
        Me.PowerSetBox.TabIndex = 13
        Me.PowerSetBox.Text = "100"
        '
        'FirePulse
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(315, 237)
        Me.Controls.Add(Me.PowerSetBox)
        Me.Controls.Add(Me.DurationSetBox)
        Me.Controls.Add(Me.PowerDisplay)
        Me.Controls.Add(Me.DurationDisplay)
        Me.Controls.Add(Me.SetValuesButton)
        Me.Controls.Add(Me.Label3)
```

```
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.CommentBox)
        Me.Controls.Add(Me.CancelPulseButton)
        Me.Controls.Add(Me.FireButton)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.MaximumSize = New System.Drawing.Size(321, 263)
        Me.MinimizeBox = False
        Me.MinimumSize = New System.Drawing.Size(321, 263)
        Me.Name = "FirePulse"
        Me.ShowIcon = False
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "Fire a Laser Pulse"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents CommentBox As System.Windows.Forms.TextBox
    Friend WithEvents CancelPulseButton As System.Windows.Forms.Button
    Friend WithEvents FireButton As System.Windows.Forms.Button
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents SetValuesButton As System.Windows.Forms.Button
    Friend WithEvents DurationDisplay As System.Windows.Forms.Label
    Friend WithEvents PowerDisplay As System.Windows.Forms.Label
    Friend WithEvents DurationSetBox As System.Windows.Forms.TextBox
    Friend WithEvents PowerSetBox As System.Windows.Forms.TextBox
End Class
```

B.3 FIREPULSE.VB

```
Public Class FirePulse
    'Code for the FirePulse form that shows when the User wants to Fire a
Manually-Issued Laser Pulse



    Private Sub FirePulse_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        'Stuff to do when the form is closing

        If e.CloseReason <> CloseReason.None Then
            e.Cancel = True
        End If

    End Sub



    Private Sub CancelPulseButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles CancelPulseButton.Click
        'User clicked Cancel

        'set the default comment
```

```
        CommentBox.Text = "(Leave a Comment)"
        CommentBox.Select(0, CommentBox.TextLength)
        CommentBox.Focus()

        DurationSetBox.Text = DurationDisplay.Text
        PowerSetBox.Text = PowerDisplay.Text

        Me.Hide()
        Application.DoEvents()

    End Sub



    Private Sub FirePulse_Shown(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Shown
        'handles stuff when the form is shown

        'set the default comment
        CommentBox.Text = "(Insert Comment Here)"
        CommentBox.Select(0, CommentBox.TextLength)
        CommentBox.Focus()

        DurationSetBox.Text = DurationDisplay.Text
        PowerSetBox.Text = PowerDisplay.Text

        Application.DoEvents()

    End Sub



    Private Sub SetValuesButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SetValuesButton.Click
        'updates the values from the set boxes

        Try

            Dim ChangedPulseDuration As Boolean = False
            Dim NewPulseDuration As Double = 0
            Dim ChangedPulsePower As Boolean = False
            Dim NewPulsePower As Double = 0
            Dim EnteredDurationValue As Double = -1
            Dim EnteredPowerValue As Double = -1

            'stop autopulsing
            RunBox.StopAutomaticPulsing()

            'test the values for convertibilty
            Try
                EnteredDurationValue = Val(DurationSetBox.Text)
            Catch TooMuchSpeed As OverflowException
                'bad value for duration
                MsgBox("Illegal Value for the new Estimated Duration:" &
vbNewLine & DurationSetBox.Text, , "Overflow")
                EnteredDurationValue = -1
            End Try

            Try
```

```
                EnteredPowerValue = Val(PowerSetBox.Text)
            Catch TooMuchSpeed As OverflowException
                'bad value for duration
                MsgBox("Illegal Value for the Next Laser Power:" & vbNewLine &
PowerSetBox.Text, , "Overflow")
                EnteredPowerValue = -1
            End Try

            'sanity check
            If EnteredDurationValue >= 0 Then
                'it works so far
                NewPulseDuration = EnteredDurationValue
                If EnteredPowerValue > 0 Then
                    'both work
                    NewPulsePower = EnteredPowerValue
                    SyncLock Windowsill.RunBoxLock
                        'load laser control values
                        If Windowsill.EstimatedLaserPulseDuration <>
NewPulseDuration Then
                            ChangedPulseDuration = True
                            Windowsill.EstimatedLaserPulseDuration =
NewPulseDuration
                        End If
                        If Windowsill.NextLaserPulsePower <> NewPulsePower Then
                            ChangedPulsePower = True
                            Windowsill.NextLaserPulsePower = NewPulsePower
                        End If
                    End SyncLock
                Else
                    MsgBox("Next Laser Pulse Power must be >0")
                End If
            Else
                MsgBox("Estimated Pulse Duration values must be >=0")
            End If

            'log the changes
            If ChangedPulseDuration = True Then
                Lumberjack.SendToLog("Estimated Laser Pulse Duration changed to
" & Str(NewPulseDuration))
            End If
            If ChangedPulsePower = True Then
                Lumberjack.SendToLog("Changed Next Laser Pulse Power to " &
Str(NewPulsePower))
            End If
            Application.DoEvents()

        Catch TooBig As OverflowException
            'something overflowed
            Lumberjack.SendToLog("Recoverable Exception in " &
Thread.CurrentThread.Name & ":  " & TooBig.Message & vbNewLine & "Details:  " &
vbNewLine & TooBig.ToString)
            MsgBox("An Overflow Exception has occurred in " &
Thread.CurrentThread.Name & ", but execution will continue.  Details:  " &
vbNewLine & TooBig.ToString, , "OUCH!")
        Catch BigException As Exception
            'something got really screwed up
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
```

```
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End Try

    End Sub



    Private Sub FireButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles FireButton.Click
        'FIRE!
        'Issues a manual-type pulse to the laser

        Dim PulsePower As Double = 0
        Dim PulseDuration As Double = 0
        Dim PulseComment As String = ""
        Dim PulseInProgress As Boolean = False

        'stop autopulsing
        RunBox.StopAutomaticPulsing()

        'get data to log
        PulseComment = CommentBox.Text
        SyncLock Windowsill.RunBoxLock
            PulsePower = Windowsill.NextLaserPulsePower
            PulseDuration = Windowsill.EstimatedLaserPulseDuration
        End SyncLock

        'prepare the values for when the box is next shown
        Me.Hide()
        PowerSetBox.Text = PowerDisplay.Text
        DurationSetBox.Text = DurationDisplay.Text
        CommentBox.Text = "(Leave a Comment)"
        CommentBox.Select(0, CommentBox.TextLength)
        CommentBox.Focus()

        'log the event
        Lumberjack.SendToLog("The User has Ordered a Manual Laser Pulse with a
Pulse Power of " & PulsePower.ToString & " and an Estimated Pulse Duration of "
& PulseDuration.ToString & " with the following Comment:  " & PulseComment)

        'Issue the pulse to the laser controller
        SyncLock Windowsill.RunBoxLock
            PulseInProgress = Windowsill.FireManualPulse
            Windowsill.FireManualPulse = True
        End SyncLock

        'log overlapping manual pulse events
        If PulseInProgress = True Then
            Lumberjack.SendToLog("Manually-Issued Laser Pulse Already in
Progress!  A Second Pulse WILL NOT Follow.")
        End If

        Application.DoEvents()
```

```
    End Sub


End Class
```

B.4 HELPBOX.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class HelpBox
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(HelpBox))
        Me.GuideTextBox = New System.Windows.Forms.TextBox
        Me.Button1 = New System.Windows.Forms.Button
        Me.SuspendLayout()
        '
        'GuideTextBox
        '
        Me.GuideTextBox.AcceptsReturn = True
        Me.GuideTextBox.AcceptsTab = True
        Me.GuideTextBox.BackColor = System.Drawing.SystemColors.Window
        Me.GuideTextBox.Font = New System.Drawing.Font("Microsoft Sans Serif",
10.0!, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
        Me.GuideTextBox.Location = New System.Drawing.Point(12, 12)
        Me.GuideTextBox.MaximumSize = New System.Drawing.Size(568, 313)
        Me.GuideTextBox.MinimumSize = New System.Drawing.Size(568, 313)
        Me.GuideTextBox.Multiline = True
        Me.GuideTextBox.Name = "GuideTextBox"
        Me.GuideTextBox.ReadOnly = True
        Me.GuideTextBox.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.GuideTextBox.Size = New System.Drawing.Size(568, 313)
        Me.GuideTextBox.TabIndex = 0
        Me.GuideTextBox.Text = "The User's Guide goes here.  " &
Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10) &
```

```
Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10) &
"Jackass."
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(505, 331)
        Me.Button1.MaximumSize = New System.Drawing.Size(75, 23)
        Me.Button1.MinimumSize = New System.Drawing.Size(75, 23)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(75, 23)
        Me.Button1.TabIndex = 1
        Me.Button1.Text = "D'oh!"
        Me.Button1.UseVisualStyleBackColor = True
        '
        'HelpBox
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(594, 375)
        Me.Controls.Add(Me.Button1)
        Me.Controls.Add(Me.GuideTextBox)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.MaximumSize = New System.Drawing.Size(600, 400)
        Me.MinimizeBox = False
        Me.MinimumSize = New System.Drawing.Size(600, 400)
        Me.Name = "HelpBox"
        Me.ShowIcon = False
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "Help! I'm stuck in a computer!"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents GuideTextBox As System.Windows.Forms.TextBox
    Friend WithEvents Button1 As System.Windows.Forms.Button
End Class
```

B.5 HELPBOX.VB

```
Public Class HelpBox
    'Code for the HelpBox form -- the form that shows when the User needs Help



    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        'Hide the help box

        Me.Hide()

    End Sub
```

```
    Private Sub HelpBox_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        'Stuff for when the box is first loaded

        Dim GuideString As String = ""

        'MsgBox("You " & vbNewLine & " IDIOT")

        'Create the head text
        GuideString = My.Application.Info.CompanyName & " " &
My.Application.Info.ProductName & " v. " & My.Application.Info.Version.Major &
"." & My.Application.Info.Version.Minor & vbNewLine
        GuideString = GuideString & "Build " &
My.Application.Info.Version.Build & ", Revision " &
My.Application.Info.Version.Revision & vbNewLine
        GuideString = GuideString &
FileVersionInfo.GetVersionInfo(Reflection.Assembly.GetExecutingAssembly().Locat
ion).OriginalFilename & vbNewLine
        GuideString = GuideString & "File Version:  " &
FileVersionInfo.GetVersionInfo(Reflection.Assembly.GetExecutingAssembly().Locat
ion).FileVersion & vbNewLine
        GuideString = GuideString & My.Application.Info.Copyright & vbNewLine &
vbNewLine & vbNewLine & vbNewLine

        'load the user's guide
        GuideString = GuideString & My.Resources.UserGuide.ToString

        'put it all in the text box
        GuideTextBox.Text = GuideString

        'set the initial location of the cursor
        GuideTextBox.SelectionStart = 0
        GuideTextBox.SelectionLength = 0

    End Sub



End Class
```

B.6 INSERTCOMMENT.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class InsertComment
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub
```

```
    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(InsertComment))
        Me.CommentOKButton = New System.Windows.Forms.Button
        Me.CommentCancelButton = New System.Windows.Forms.Button
        Me.CommentBox = New System.Windows.Forms.TextBox
        Me.SuspendLayout()
        '
        'CommentOKButton
        '
        Me.CommentOKButton.Location = New System.Drawing.Point(12, 231)
        Me.CommentOKButton.MaximumSize = New System.Drawing.Size(75, 23)
        Me.CommentOKButton.MinimumSize = New System.Drawing.Size(75, 23)
        Me.CommentOKButton.Name = "CommentOKButton"
        Me.CommentOKButton.Size = New System.Drawing.Size(75, 23)
        Me.CommentOKButton.TabIndex = 0
        Me.CommentOKButton.Text = "OK"
        Me.CommentOKButton.UseVisualStyleBackColor = True
        '
        'CommentCancelButton
        '
        Me.CommentCancelButton.Location = New System.Drawing.Point(205, 231)
        Me.CommentCancelButton.MaximumSize = New System.Drawing.Size(75, 23)
        Me.CommentCancelButton.MinimumSize = New System.Drawing.Size(75, 23)
        Me.CommentCancelButton.Name = "CommentCancelButton"
        Me.CommentCancelButton.Size = New System.Drawing.Size(75, 23)
        Me.CommentCancelButton.TabIndex = 1
        Me.CommentCancelButton.Text = "Cancel"
        Me.CommentCancelButton.UseVisualStyleBackColor = True
        '
        'CommentBox
        '
        Me.CommentBox.Location = New System.Drawing.Point(12, 12)
        Me.CommentBox.MaximumSize = New System.Drawing.Size(268, 213)
        Me.CommentBox.MinimumSize = New System.Drawing.Size(268, 213)
        Me.CommentBox.Multiline = True
        Me.CommentBox.Name = "CommentBox"
        Me.CommentBox.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.CommentBox.Size = New System.Drawing.Size(268, 213)
        Me.CommentBox.TabIndex = 2
        '
        'InsertComment
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(294, 275)
        Me.Controls.Add(Me.CommentBox)
        Me.Controls.Add(Me.CommentCancelButton)
        Me.Controls.Add(Me.CommentOKButton)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
```

```
        Me.MaximizeBox = False
        Me.MaximumSize = New System.Drawing.Size(300, 300)
        Me.MinimizeBox = False
        Me.MinimumSize = New System.Drawing.Size(300, 300)
        Me.Name = "InsertComment"
        Me.ShowIcon = False
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "Insert a Comment"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents CommentOKButton As System.Windows.Forms.Button
    Friend WithEvents CommentCancelButton As System.Windows.Forms.Button
    Friend WithEvents CommentBox As System.Windows.Forms.TextBox
End Class
```

B.7 INSERTCOMMENT.VB

```
Public Class InsertComment
    'Code for the InsertComment form that shows when the User wants to insert a
Comment into the Log



    Private Sub CommentCancelButton_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CommentCancelButton.Click
        'User Clicked Cancel Button

        'set the default comment
        CommentBox.Text = "(Insert Comment Here)"
        CommentBox.Select(0, CommentBox.TextLength)
        CommentBox.Focus()
        Me.Hide()

    End Sub



    Private Sub CommentOKButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CommentOKButton.Click
        'User Clicked OK button

        'Send the Comment to the Log
        Lumberjack.SendToLog("User Comment:  " & CommentBox.Text)
        'set the default comment
        CommentBox.Text = "(Insert Comment Here)"
        CommentBox.Select(0, CommentBox.TextLength)
        CommentBox.Focus()
        Me.Hide()

    End Sub



    Private Sub InsertComment_Shown(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Shown
```

```
        'Form is Shown

        'set the default comment
        CommentBox.Text = "(Insert Comment Here)"
        CommentBox.Select(0, CommentBox.TextLength)
        CommentBox.Focus()
        Application.DoEvents()

    End Sub


End Class
```

B.8 LONGSHOREMAN.VB

```
Option Explicit On
Option Strict On

Friend NotInheritable Class Longshoreman

    'This class deals with managing the second serial port
    'Its thread only runs if there is a second port in use
    'However, it maintains the Laser Control Mode variable for the program
    '
    '
    Friend Shared LaserPorter As New Thread(AddressOf
Longshoreman.SecondPortManager)
    Friend Shared LaserControlMode As Long = 0
    Friend Shared WithEvents COMPort2 As New System.IO.Ports.SerialPort
    Friend Shared LaserPortLock As New Object



    Private Shared Sub SecondPortManager()
        'Only used in dual-port mode
        'in dual-port mode, the laser is connected to a second serial port
        'instead of piggy-backing on the speed controller port

        Dim LocalExecutionStatus As Long
        Dim LocalProgramExit As Boolean
        Dim Proceed As Boolean
        Dim LocalPortName As String
        Dim LocalPortSpeed As Integer
        Dim LocalPortDataBits As Integer
        Dim LocalPortStopBits As System.IO.Ports.StopBits
        Dim LocalPortFlowControl As System.IO.Ports.Handshake
        Dim LocalPortEncoding As System.Text.Encoding
        Dim LocalPortOpen As Boolean = False
        Dim LocalRXString As String = ""
        Dim LocalTXString As String = ""
        Dim LocalRXChar(0) As Char
        Dim CharactersRead As Integer = 0
        Dim CharactersRemain As Boolean = False
        Dim PreviousCharacterNewline As Boolean = False
        Dim LastRXTimeTicks As Long = 0
        Dim RXTimeoutToLogTicks As Long = 2500000
```

```
        Dim BytesToSend() As Byte
        Dim NumberBytes As Integer = 0
        Dim SendRefObj As Object = CObj(0)
        Dim ShortString As String = ""
        Dim FoundLimit As Boolean = False
        Dim SendLength As Integer = 0
        Dim WriteNumber As Long = 0
        Dim ClearForLaser As Boolean = False
        Dim LaserEOL As String = ""
        Dim RecentTX As Boolean = False

        Try
            'much of this mirrors stuff that happens elsewhere in other Program
modes
            Thread.CurrentThread.Name = "LaserPorter"
            Proceed = False
            LocalRXChar(0) = CChar("")
            'wait for the speed control port to open
            Do
                'get speed control port status
                SyncLock Windowsill.RunBoxLock
                    Proceed = Windowsill.SpeedControlReady
                End SyncLock
                'check for program exit condition
                SyncLock CentralClass.ProgramExitLock
                    LocalProgramExit = CentralClass.ExitProgram
                End SyncLock
                If LocalProgramExit = True Then Proceed = True
                Application.DoEvents()
                System.Threading.Thread.Sleep(3500)
            Loop While Proceed = False

            'get port settings and try to open the port
            SyncLock CentralClass.FileOpsLock
                LocalPortName = CentralClass.LaserPort
                LocalPortSpeed = CentralClass.ComPortBaudRate
                LocalPortDataBits = CentralClass.ComPortDataBits
                LocalPortStopBits = CentralClass.ComPortStopBits
                LocalPortFlowControl = CentralClass.ComPortFlowControl
                LocalPortEncoding = CentralClass.ComPortEncoding
                LaserEOL = CentralClass.LaserNewlineString
            End SyncLock

            Proceed = False
            'open the port
            Do

                Try
                    'log it
                    Lumberjack.SendToLog("Attempting to open Laser Port (" &
LocalPortName & ") with the following settings:  " & LocalPortSpeed.ToString &
" baud, " & LocalPortDataBits.ToString & " data bits, " &
LocalPortStopBits.ToString & " stop bits, flow control=" &
LocalPortFlowControl.ToString & ", and " & LocalPortEncoding.ToString & "
Encoding.")
                    'set up and open the port
                    SyncLock Longshoreman.LaserPortLock
                        Longshoreman.COMPort2.PortName = LocalPortName
                        Longshoreman.COMPort2.BaudRate = LocalPortSpeed
```

```
                        Longshoreman.COMPort2.DataBits = LocalPortDataBits
                        Longshoreman.COMPort2.StopBits = LocalPortStopBits
                        Longshoreman.COMPort2.Handshake = LocalPortFlowControl
                        Longshoreman.COMPort2.Encoding = LocalPortEncoding
                        'set the read and write buffer size in bytes
                        Longshoreman.COMPort2.ReadBufferSize = 16384
                        Longshoreman.COMPort2.WriteBufferSize = 16384
                        'open the port
                        Longshoreman.COMPort2.Open()
                        LocalPortOpen = Longshoreman.COMPort2.IsOpen
                    End SyncLock
                    'log the results
                    If LocalPortOpen = True Then
                        'success
                        Lumberjack.SendToLog("Success in Opening the Laser Port
(" & LocalPortName & ")")
                        'proceed with the program
                        Proceed = True
                    Else
                        'some failure without an exception getting caught
                        Lumberjack.SendToLog("Failure in Opening the Laser Port
(" & LocalPortName & ")")
                        'see if user wants to try again
                        If MsgBox("Failure in attempt to open the laser port "
& LocalPortName & vbNewLine & "Retry?", MsgBoxStyle.YesNo, "Laser Port
Problem") = MsgBoxResult.Yes Then
                            'user wants to retry
                            Lumberjack.SendToLog("Will retry opening laser
port")
                        Else
                            'user wants to abort
                            Proceed = True
                            Lumberjack.SendToLog("Abort opening laser port,
ending program")
                            SyncLock CentralClass.ProgramExitLock
                                CentralClass.ExitProgram = True
                            End SyncLock
                        End If
                    End If

                Catch SerialException As Exception
                    Lumberjack.SendToLog("Exception in attempt to open the
laser port " & LocalPortName & ":  " & vbNewLine & SerialException.Message)
                    If MsgBox("Exception in attempt to open the laser port " &
LocalPortName & ":  " & vbNewLine & SerialException.Message & vbNewLine &
"Retry?", MsgBoxStyle.YesNo, "Laser Port Problem") = MsgBoxResult.Yes Then
                        'user wants to retry
                        Lumberjack.SendToLog("Will retry opening laser port")
                    Else
                        'user wants to abort
                        Proceed = True
                        Lumberjack.SendToLog("Abort opening laser port, ending
program")
                        SyncLock CentralClass.ProgramExitLock
                            CentralClass.ExitProgram = True
                        End SyncLock
                    End If

                End Try
```

```
                'Check to see if the program should exit
                SyncLock CentralClass.ProgramExitLock
                    If CentralClass.ExitProgram = True Then
                        Proceed = True
                        LocalProgramExit = True
                    End If
                End SyncLock
            Loop While Proceed = False

            'signal that the program is ready
            SyncLock Windowsill.RunBoxLock
                Windowsill.LaserPortReady = True
            End SyncLock

            'main part of the program
            'it doesn't care if the program is in early shutdown or pure
operation
            'so it does the same loop until the correct shutdown stage is
reached
            Do
                SyncLock CentralClass.ProgramExitLock
                    LocalExecutionStatus = CentralClass.ExecutionStage
                End SyncLock
                System.Threading.Thread.Sleep(65)
                Application.DoEvents()
                '
                'get data from the port
                SyncLock Longshoreman.LaserPortLock
                    LocalPortOpen = Longshoreman.COMPort2.IsOpen
                End SyncLock

                If LocalPortOpen = True Then
                    'grab the characters buffered by the serialport one at a
time
                    Do
                        'read one character from the port, if there are any
                        SyncLock Longshoreman.LaserPortLock
                            If Longshoreman.COMPort2.BytesToRead > 0 Then
                                CharactersRead =
Longshoreman.COMPort2.Read(LocalRXChar, 0, 1)
                                'remember the last rx read time
                                LastRXTimeTicks = Now.Ticks
                            Else
                                'no characters read
                                CharactersRead = 0
                            End If
                        End SyncLock
                        If CharactersRead > 0 Then
                            'characters were actually read, process the data
                            'check for cr or lf
                            If (LocalRXChar(0).ToString = vbCr) Or
(LocalRXChar(0).ToString = vbLf) Then
                                'current character is a newline sort of thing
                                If PreviousCharacterNewline = True Then
                                    'previous character was a newline sort of
character
                                    'so is the current line
```

```
                                    'log it all and clear the
previouscharacternewline condition
                                    PreviousCharacterNewline = False
                                    LocalRXString = LocalRXString &
LocalRXChar(0).ToString
                                    'send it all to the runbox
                                    SyncLock Windowsill.RunBoxLock
                                        Windowsill.LaserRX = Windowsill.LaserRX
& LocalRXString
                                    End SyncLock
                                    'log it
                                    Lumberjack.SendToLog("RX From Laser:  " &
LocalRXString)
                                    'clear the string
                                    LocalRXString = ""
                                Else
                                    'previous character was NOT a newline
character
                                    'but the current line is
                                    'append it to the local rx string, but wait
before logging
                                    PreviousCharacterNewline = True
                                    LocalRXString = LocalRXString &
LocalRXChar(0)
                                End If
                            ElseIf PreviousCharacterNewline = True Then
                                'previous but not current character was a
newline
                                'send the previous data where they all belong
                                'send it all to the runbox
                                SyncLock Windowsill.RunBoxLock
                                    Windowsill.LaserRX = Windowsill.LaserRX &
LocalRXString
                                End SyncLock
                                'log it
                                Lumberjack.SendToLog("RX From Laser:  " &
LocalRXString)
                                'clear the local rx string and put the current
character in it
                                LocalRXString = LocalRXChar(0).ToString
                                PreviousCharacterNewline = False
                            Else
                                'neither the current nor the last character
were newlines
                                'append the received character to the local rx
string
                                LocalRXString = LocalRXString &
LocalRXChar(0).ToString
                            End If
                            '
                        End If

                        'check for remaining characters
                        SyncLock Longshoreman.LaserPortLock
                            If Longshoreman.COMPort2.BytesToRead > 0 Then
                                'data remains
                                CharactersRemain = True
                            Else
                                CharactersRemain = False
```

```
                            'no data left
                        End If
                    End SyncLock
                Loop While CharactersRemain = True
            End If

            'check for time to dump the rx data to the log
            'only if it hasn't received anything in the timeout period
            If (Now.Ticks - LastRXTimeTicks) > RXTimeoutToLogTicks Then
                'last rx time was more than one timeout ago
                'if there's data to log, log it
                If LocalRXString <> "" Then
                    'stuff to log
                    'send it all to the runbox
                    SyncLock Windowsill.RunBoxLock
                        Windowsill.LaserRX = Windowsill.LaserRX &
LocalRXString
                    End SyncLock
                    'log it
                    Lumberjack.SendToLog("RX From Laser:  " &
LocalRXString)
                    'clear the string
                    LocalRXString = ""
                End If
            End If

            'write block

            'tell fire control to hold off momentarily
            SyncLock Stevedore.SpeedPortLock
                Stevedore.ClearToFireLaser = False
            End SyncLock

            'send data to port here
            'get any data
            RecentTX = False
            SyncLock Windowsill.RunBoxLock
                LocalTXString = LocalTXString & Windowsill.CommandToLaser
                'got the data, clear the shared variable
                Windowsill.CommandToLaser = ""
            End SyncLock
            'only send when the output buffer is empty to ensure it goes in
the correct order
            If LocalTXString <> "" Then
                'set the transmission flag for later use
                RecentTX = True
                'check for an empty buffer
                SyncLock Longshoreman.LaserPortLock
                    If Longshoreman.COMPort2.BytesToWrite = 0 Then
                        'empty buffer, write data
                        'localportencoding
                        If LocalPortEncoding.GetByteCount(LocalTXString) >
Longshoreman.COMPort2.WriteBufferSize Then
                            'too many characters
                            'split the write in half in a loop until it
fits
                            FoundLimit = False
                            SendLength = LocalTXString.Length
                            Do
```

```
                                    SendLength = CInt(SendLength / 2)
                                    'see if the max bytes from that can be
bufferred
                                    If
LocalPortEncoding.GetMaxByteCount(SendLength) <
Longshoreman.COMPort2.WriteBufferSize Then
                                        'it works
                                        FoundLimit = True
                                    ElseIf SendLength < 10 Then
                                        'it should be longer, something is
really wrong
                                        'proceed anyway
                                        FoundLimit = True
                                    End If
                                Loop While FoundLimit = False
                                'get the string to send
                                ShortString = LocalTXString.Substring(0,
SendLength)
                                'and remove it from the local tx string
                                LocalTXString = LocalTXString.Remove(0,
SendLength)
                                'encode the string
                                BytesToSend =
LocalPortEncoding.GetBytes(ShortString)
                                NumberBytes = BytesToSend.Length
                                'now it's ready to write
                            Else
                                'everything can fit in the buffer immediately
                                'encode the string
                                BytesToSend =
LocalPortEncoding.GetBytes(LocalTXString)
                                NumberBytes = BytesToSend.Length
                                'clear the local string
                                LocalTXString = ""
                                'ready to send
                            End If
                            'write it to the serial port output asynchronously
                            SendRefObj = CObj(WriteNumber)

Longshoreman.COMPort2.BaseStream.BeginWrite(BytesToSend, 0, NumberBytes,
AddressOf Longshoreman.FinishedComPort2Send, SendRefObj)
                            WriteNumber += 1
                        End If
                    End SyncLock
                End If

                'Update Laser Fire Control
                ClearForLaser = True
                'check for request to fire
                If ClearForLaser = True Then
                    SyncLock Windowsill.RunBoxLock
                        If Windowsill.LaserTXDataCritical = False Then
                            'laser not in firing sequence, laser is not clear
to fire
                            ClearForLaser = False
                        End If
                    End SyncLock
                End If
                'check saved data
```

```
                If LocalTXString <> "" Then
                    'there's saved data to be transmitted, laser is not clear
to fire
                    ClearForLaser = False
                End If
                'check available data
                If ClearForLaser = True Then
                    SyncLock Windowsill.RunBoxLock
                        If Windowsill.CommandToLaser <> "" Then
                            'there's data left to be loaded and transmitted,
laser is not clear to fire
                            ClearForLaser = False
                        End If
                    End SyncLock
                End If
                'check bufferred data
                If ClearForLaser = True Then
                    SyncLock Longshoreman.LaserPortLock
                        If Longshoreman.COMPort2.BytesToWrite <> 0 Then
                            'there's bufferred data in transmission, laser is
not clear to fire
                            ClearForLaser = False
                        End If
                    End SyncLock
                End If
                If RecentTX = True Then
                    'don't signal a clear line just yet
                    ClearForLaser = False
                End If
                'send value to fire control
                SyncLock Stevedore.SpeedPortLock
                    Stevedore.ClearToFireLaser = ClearForLaser
                End SyncLock

            Loop While LocalExecutionStatus < 1020

            'signal fire control to that the laser is not clear to fire
            SyncLock Stevedore.SpeedPortLock
                Stevedore.ClearToFireLaser = False
            End SyncLock

            'close the port
            Try
                'if the port was opened, close it
                If LocalPortOpen = True Then
                    'log it
                    Lumberjack.SendToLog("Attempting to Close the Laser Port ("
& LocalPortName & ")")
                    'send final string to the laser port, if there is one
                    'get any data
                    SyncLock Windowsill.RunBoxLock
                        LocalTXString = LocalTXString &
Windowsill.CommandToLaser
                        Windowsill.CommandToLaser = ""
                    End SyncLock
                    'transmit any data
                    LocalTXString = LocalTXString & LaserEOL

                    Try
```

```
                        'perform the write
                        SyncLock Longshoreman.LaserPortLock
                            Longshoreman.COMPort2.Write(LocalTXString)
                        End SyncLock
                    Catch FinishedWriteException As Exception
                        'log the issue
                        Lumberjack.SendToLog("Exception in final write on " &
LocalPortName & ":  " & FinishedWriteException.Message)
                    End Try

                    System.Threading.Thread.Sleep(250)
                    'get all the data from the port, then close it
                    SyncLock Longshoreman.LaserPortLock
                        If Longshoreman.COMPort2.IsOpen = True Then
                            LocalRXString = LocalRXString &
Longshoreman.COMPort2.ReadExisting()
                            'the port is open, close it
                            Longshoreman.COMPort2.Close()
                            Longshoreman.COMPort2.Dispose()
                        End If
                        LocalPortOpen = Longshoreman.COMPort2.IsOpen
                    End SyncLock
                    System.Threading.Thread.Sleep(1000)
                    'log success/failure to close the port
                    If LocalPortOpen = True Then
                        'port is still open, log an error
                        Lumberjack.SendToLog("Laser Port (" & LocalPortName &
") did not close successfully")
                    Else
                        'port closed
                        Lumberjack.SendToLog("Successfully Closed the Laser
Port (" & LocalPortName & ")")
                    End If
                End If

            Catch SerialPortClosureProblem As Exception
                'log it
                Lumberjack.SendToLog("Exception while attempting to close the
Laser Port (" & LocalPortName & "):  " & vbNewLine &
SerialPortClosureProblem.Message)
            End Try

            'send the output where it belongs
            If LocalRXString <> "" Then
                'log the final rx string
                Lumberjack.SendToLog("Final Text Received from the Laser:  " &
LocalRXString)
                'mirror it on the runbox
                SyncLock Windowsill.RunBoxLock
                    Windowsill.LaserRX = Windowsill.LaserRX & LocalRXString
                End SyncLock
                'clear the string
                LocalRXString = ""
            End If

            System.Threading.Thread.Sleep(3000)
            'log termination
            Lumberjack.SendToLog("Laser Port Module Stopping")
```

```
        Catch ThreadNeededKilling As ThreadAbortException
            'the thread was aborted
            Application.ExitThread()
        Catch BigException As Exception
            'something got really screwed up
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End Try

        Application.ExitThread()

    End Sub



    Private Shared Sub FinishedComPort2Send(ByVal TXResult As
System.IAsyncResult)
        'gets called in asynchronous writes to the laser port
        'when they finish transmitting data

        Try
            SyncLock Longshoreman.LaserPortLock
                'end the (completed) write
                Longshoreman.COMPort2.BaseStream.EndWrite(TXResult)
            End SyncLock
        Catch ex As Exception
            'something went wrong
            'log the error
            Lumberjack.SendToLog("Exception in the result of transmission to
laser:  " & ex.Message)
            Lumberjack.SendToLog("Exception Data:  " &
TXResult.AsyncState.ToString())
        End Try

    End Sub



End Class
```

B.9 LUMBERJACK.VB

```
Option Explicit On
Option Strict On

Friend NotInheritable Class Lumberjack

    'This class contains the logging system
    'It executes in a thread, here the BostonLogger
```

```
    'Before the thread is launched, the OutputLogFile needs to be opened for
writing
    'This must be done by an outside routine, not here
    '
    '
    Friend Shared BostonLogger As New Thread(AddressOf Lumberjack.LogWriter)
    Friend Shared OutputLogFile As System.IO.TextWriter
    Friend Shared LogFileLock As New Object
    Private Shared NextOutputString As String = ""
    Private Shared TimberLock As New Object
    Private Shared LogDividerString As String =
"================================"



    Private Shared Function BannerInfo() As String
        'hands the startup info to the logger

        Dim NextString As String = ""
        Dim DividerString As String = Lumberjack.LogDividerString

        SyncLock CentralClass.FileOpsLock
            'startup banner
            NextString = vbNewLine & vbNewLine & DividerString & vbNewLine
            NextString = NextString & My.Application.Info.CompanyName & " " &
My.Application.Info.ProductName & " v. " & My.Application.Info.Version.Major &
"." & My.Application.Info.Version.Minor & vbNewLine
            NextString = NextString & "Build " &
My.Application.Info.Version.Build & ", Revision " &
My.Application.Info.Version.Revision & vbNewLine
            NextString = NextString &
FileVersionInfo.GetVersionInfo(Reflection.Assembly.GetExecutingAssembly().Locat
ion).OriginalFilename & " File Version:  " &
FileVersionInfo.GetVersionInfo(Reflection.Assembly.GetExecutingAssembly().Locat
ion).FileVersion & vbNewLine
            NextString = NextString & DividerString & vbNewLine
            'give the filename
            NextString = NextString & "Log File:  " & vbNewLine &
System.IO.Path.GetFullPath(CentralClass.Filename) & vbNewLine
            'give append/replace flag
            If CentralClass.AppendFile = True Then
                NextString = NextString & "Appending any existing file." &
vbNewLine
            Else
                NextString = NextString & "Replacing any existing file." &
vbNewLine
            End If
            'reflect status of advanced timing features
            NextString = NextString & "Advanced Timing Features:  "
            If CentralClass.AdvancedTiming = True Then
                NextString = NextString & "Enabled" & vbNewLine
            Else
                NextString = NextString & "Disabled" & vbNewLine
            End If
            'give selected ports, speed, and start time
            NextString = NextString & "Speed Controller Port:  " &
CentralClass.SpeedPort & vbNewLine
            NextString = NextString & "Laser Port:  " & CentralClass.LaserPort
& vbNewLine
```

```
            NextString = NextString & "Port Speed:  " &
CentralClass.ComPortBaudRate & " baud" & vbNewLine
            NextString = NextString & "Program Startup:  " &
CentralClass.ProgramStartTime & vbNewLine
            NextString = NextString & DividerString
            NextString = NextString & Lumberjack.LogStamp & "Begin Logging"
        End SyncLock

        BannerInfo = NextString

    End Function



    Private Shared Sub SawmillEcho(ByVal Logged As String)

        'This directs the data logged back out to the runbox mirror

        SyncLock Windowsill.RunBoxLock
            'echo the data to the RunBox
            Windowsill.LogWritten = Windowsill.LogWritten & Logged
        End SyncLock

    End Sub



    Private Shared Function LogStamp() As String

        'gives a new line and the formatted time stamp for the log

        LogStamp = vbNewLine & "[" & Format(Now(), "MM/dd/yyyy HH:mm:ss.ff") &
"]:  "

    End Function



    Public Shared Sub SendToLog(ByVal ToLog As String)

        'sends the input string to the log file
        SyncLock Lumberjack.TimberLock
            Lumberjack.NextOutputString = Lumberjack.NextOutputString &
LogStamp() & ToLog
        End SyncLock

        'evidently the thread.interrupt method is dangerous, so it's commented
out here

        'Try
        '    'if the logger is asleep, wake it
        '    If Lumberjack.BostonLogger.ThreadState =
Threading.ThreadState.WaitSleepJoin Then
        '        Lumberjack.BostonLogger.Interrupt()
        '    End If
        'Catch SomethingUnexpected As Exception
        '    Beep()
        '    'No exceptions are actually expected to ever appear here
        '    'If they do, they probably can't be logged
```

```
        'End Try

    End Sub



    Private Shared Sub LogWriter()
        'this is meant to run in its own thread and send the log queue to the
log file
        'it echoes it to the runbox form as well through the SawmillEcho sub

        Dim NextString As String = ""
        Dim LocalProgramExit As Boolean = False
        Dim DeltaTicks As Long
        Dim LastFlush As Long
        Dim LocalExecutionStage As Long
        Dim LocalTimeoutCounter As Long
        Dim LocalTimeoutDelay As Long
        Dim MaxCharacterTransfer As Integer = 32767
        Dim CharactersToTransfer As Integer = 0
        Dim DividerString As String = Lumberjack.LogDividerString

        'trap otherwise unhandled exceptions, kill the thread if any pop up
        Try
            'startup stuff
            Thread.CurrentThread.Name = "Boston_Logger"
            'set the flush frequency to ~10 seconds
            DeltaTicks = 100000000
            'set timeout to ~1000 seconds
            LocalTimeoutDelay = 10000000000
            NextString = BannerInfo()
            SyncLock Lumberjack.TimberLock
                'purge the next output string, if it already has data
                If Lumberjack.NextOutputString <> "" Then
                    'don't get more characters than the i/o can handle
                    If NextString.Length + Lumberjack.NextOutputString.Length >
MaxCharacterTransfer Then
                        'too long for a single operation
                        'only do one, hold the rest over for the main loop
                        'next if-then is really only a formality
                        If NextString.Length < MaxCharacterTransfer Then
                            'we can append it, so do so
                            CharactersToTransfer = MaxCharacterTransfer -
NextString.Length
                            NextString = NextString &
Lumberjack.NextOutputString.Substring(0, CharactersToTransfer)
                            'remove the transferred characters
                            Lumberjack.NextOutputString =
Lumberjack.NextOutputString.Remove(0, CharactersToTransfer)
                        End If
                    Else
                        'short enough for a single operation
                        NextString = NextString & Lumberjack.NextOutputString
                        Lumberjack.NextOutputString = ""
                    End If
                End If
            End SyncLock

            'now write the startup material to the log file
```

```
            SyncLock Lumberjack.LogFileLock
                Try
                    Lumberjack.OutputLogFile.Write(NextString)
                    LastFlush = Now.Ticks
                    Lumberjack.OutputLogFile.Flush()
                Catch ex As Exception
                    MsgBox("An exception has occurred in " &
Thread.CurrentThread.Name & " and the program will shut down" & vbNewLine &
ex.Message, , "OUCH!")
                    SyncLock CentralClass.ProgramExitLock
                        CentralClass.ExitProgram = True
                    End SyncLock
                    SyncLock Lumberjack.TimberLock
                        Lumberjack.NextOutputString =
Lumberjack.NextOutputString & Lumberjack.LogStamp & "System Error:  " &
ex.Message
                    End SyncLock
                End Try
            End SyncLock

            'copy the initial logging to the runbox
            Lumberjack.SawmillEcho(NextString)

            'clear the output string
            NextString = ""

            'now loop for the main operation
            Do
                'check for new data
                SyncLock Lumberjack.TimberLock

                    'check for new data
                    If Lumberjack.NextOutputString <> "" Then
                        'don't get more characters than the i/o can handle
                        If Lumberjack.NextOutputString.Length >
MaxCharacterTransfer Then
                            'too long for a single operation
                            'only do one, leave the rest
                            CharactersToTransfer = MaxCharacterTransfer
                            NextString =
Lumberjack.NextOutputString.Substring(0, CharactersToTransfer)
                            'remove the transferred characters
                            Lumberjack.NextOutputString =
Lumberjack.NextOutputString.Remove(0, CharactersToTransfer)
                        Else
                            'short enough for a single operation
                            NextString = NextString &
Lumberjack.NextOutputString
                            Lumberjack.NextOutputString = ""
                        End If
                    End If
                End SyncLock

                'write any new data to the disk and RunBox
                If NextString <> "" Then
                    SyncLock Lumberjack.LogFileLock
                        'write the data
                        Try
                            Lumberjack.OutputLogFile.Write(NextString)
```

```
                            'check to see if it's time to flush the cache
                            If (Now.Ticks - LastFlush) > DeltaTicks Then
                                'time to flush
                                LastFlush = Now.Ticks
                                Lumberjack.OutputLogFile.Flush()
                            End If
                        Catch ex As Exception
                            MsgBox("An exception has occurred in " &
Thread.CurrentThread.Name & " and the program will shut down" & vbNewLine &
ex.Message, , "OUCH!")
                            SyncLock CentralClass.ProgramExitLock
                                CentralClass.ExitProgram = True
                            End SyncLock
                            SyncLock Lumberjack.TimberLock
                                Lumberjack.NextOutputString =
Lumberjack.NextOutputString & Lumberjack.LogStamp & "System Error:  " &
ex.Message
                            End SyncLock
                        End Try
                    End SyncLock
                    Lumberjack.SawmillEcho(NextString)
                    'clear the string
                    NextString = ""
                End If

                'wait
                Try
                    'this was 55 ms, but that expected a thread.interrupt
method, which has turned out to be dangerous
                    'so the sleep period was shortened instead
                    System.Threading.Thread.Sleep(35)
                Catch ex As ThreadInterruptedException
                    'sleep was interrupted
                    Exit Try
                End Try
                Application.DoEvents()
                'get exit status
                SyncLock CentralClass.ProgramExitLock
                    LocalProgramExit = CentralClass.ExitProgram
                End SyncLock

            Loop While LocalProgramExit = False

            'put shutdown matter here
            LocalTimeoutCounter = Now.Ticks

            Do
                'wait for the correct time to shut all the way down
                'in the meantime, continue to log, but do not echo to runbox if
its exit level has been reached
                'check for new data
                SyncLock Lumberjack.TimberLock
                    'check for new data
                    If Lumberjack.NextOutputString <> "" Then
                        'don't get more characters than the i/o can handle
                        If Lumberjack.NextOutputString.Length >
MaxCharacterTransfer Then
                            'too long for a single operation
                            'only do one, leave the rest
```

```
                            CharactersToTransfer = MaxCharacterTransfer
                            NextString =
Lumberjack.NextOutputString.Substring(0, CharactersToTransfer)
                            'remove the transferred characters
                            Lumberjack.NextOutputString =
Lumberjack.NextOutputString.Remove(0, CharactersToTransfer)
                        Else
                            'short enough for a single operation
                            NextString = NextString &
Lumberjack.NextOutputString
                            Lumberjack.NextOutputString = ""
                        End If
                    End If
                End SyncLock
                'write any new data to the disk
                If NextString <> "" Then
                    SyncLock Lumberjack.LogFileLock
                        'write the data
                        Try
                            Lumberjack.OutputLogFile.Write(NextString)
                        Catch ex As Exception
                            MsgBox("An exception has occurred in " &
Thread.CurrentThread.Name & " and the program will shut down" & vbNewLine &
ex.Message, , "OUCH!")
                            SyncLock CentralClass.ProgramExitLock
                                CentralClass.ExitProgram = True
                            End SyncLock
                            SyncLock Lumberjack.TimberLock
                                Lumberjack.NextOutputString =
Lumberjack.NextOutputString & Lumberjack.LogStamp & "System Error:  " &
ex.Message
                            End SyncLock
                        End Try
                    End SyncLock
                    If LocalExecutionStage < 1050 Then
                        'User Interface should still be running
                        Lumberjack.SawmillEcho(NextString)
                    End If
                    'clear the string
                    NextString = ""
                End If
                Try
                    System.Threading.Thread.Sleep(55)
                Catch ex As ThreadInterruptedException
                    'sleep was interrupted
                    Exit Try
                End Try
                Application.DoEvents()
                SyncLock CentralClass.ProgramExitLock
                    'get the program execution stage
                    LocalExecutionStage = CentralClass.ExecutionStage
                End SyncLock
                'nominally, loop until the proper shutdown stage is reached
before exiting
                'but if it takes too long then exit after a timeout
            Loop While ((Now.Ticks - LocalTimeoutCounter) < LocalTimeoutDelay)
And (LocalExecutionStage < 1100)

            'Final Shutdown Stage for Logging
```

```
            'proceed to exit
            Do
                'loop until all the log material is gathered

                'check for new data
                SyncLock Lumberjack.TimberLock
                    'check for new data
                    If Lumberjack.NextOutputString <> "" Then
                        'don't get more characters than the i/o can handle
                        If Lumberjack.NextOutputString.Length >
MaxCharacterTransfer Then
                            'too long for a single operation
                            'only do one, leave the rest
                            CharactersToTransfer = MaxCharacterTransfer
                            NextString =
Lumberjack.NextOutputString.Substring(0, CharactersToTransfer)
                            'remove the transferred characters
                            Lumberjack.NextOutputString =
Lumberjack.NextOutputString.Remove(0, CharactersToTransfer)
                        Else
                            'short enough for a single operation
                            NextString = NextString &
Lumberjack.NextOutputString
                            Lumberjack.NextOutputString = ""
                        End If
                    End If
                End SyncLock

                'write gathered data
                If NextString <> "" Then
                    SyncLock Lumberjack.LogFileLock
                        'write the data
                        Try
                            Lumberjack.OutputLogFile.Write(NextString)
                        Catch ex As Exception
                            MsgBox("An exception has occurred in " &
Thread.CurrentThread.Name & " and the program will shut down" & vbNewLine &
ex.Message, , "OUCH!")
                            SyncLock CentralClass.ProgramExitLock
                                CentralClass.ExitProgram = True
                            End SyncLock
                            SyncLock Lumberjack.TimberLock
                                Lumberjack.NextOutputString =
Lumberjack.NextOutputString & Lumberjack.LogStamp & "System Error:  " &
ex.Message
                            End SyncLock
                        End Try
                    End SyncLock
                    'clear the string
                    NextString = ""
                End If

                'see if there's additional data
                SyncLock Lumberjack.TimberLock
                    'get number of remaining characters
                    CharactersToTransfer = Lumberjack.NextOutputString.Length
                End SyncLock

            Loop While CharactersToTransfer > 0
```

```
            NextString = NextString & Lumberjack.LogStamp & "End Logging" &
vbNewLine & DividerString & vbNewLine
            'write any new data to the disk
            If NextString <> "" Then
                SyncLock Lumberjack.LogFileLock
                    'write the data
                    Try
                        Lumberjack.OutputLogFile.Write(NextString)
                    Catch ex As Exception
                        MsgBox("An exception has occurred in " &
Thread.CurrentThread.Name & " and the program will shut down" & vbNewLine &
ex.Message, , "OUCH!")
                        SyncLock CentralClass.ProgramExitLock
                            CentralClass.ExitProgram = True
                        End SyncLock
                        SyncLock Lumberjack.TimberLock
                            Lumberjack.NextOutputString =
Lumberjack.NextOutputString & Lumberjack.LogStamp & "System Error:  " &
ex.Message
                        End SyncLock
                    End Try
                End SyncLock
                'clear the string
                NextString = ""
            End If
            SyncLock Lumberjack.LogFileLock
                Try
                    'time to flush
                    Lumberjack.OutputLogFile.Flush()
                    'close the file
                    Lumberjack.OutputLogFile.Close()
                Catch ex As Exception
                    MsgBox("An exception has occurred in " &
Thread.CurrentThread.Name & " and the program will shut down" & vbNewLine &
ex.Message, , "OUCH!")
                    SyncLock CentralClass.ProgramExitLock
                        CentralClass.ExitProgram = True
                    End SyncLock
                    SyncLock Lumberjack.TimberLock
                        Lumberjack.NextOutputString =
Lumberjack.NextOutputString & Lumberjack.LogStamp & "System Error:  " &
ex.Message
                    End SyncLock
                End Try
            End SyncLock

        Catch ThreadNeededKilling As ThreadAbortException
            'the thread was aborted
            Application.ExitThread()
        Catch BigException As Exception
            'something got really screwed up
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End Try
```

```
        Application.ExitThread()

    End Sub


End Class
```

B.10 MAINCODE.VB

```
Option Explicit On
Option Strict On

'Imports System.Security.AccessControl



Module MainCode

    Public Declare Function QueryPerformanceCounter Lib "kernel32" (ByRef
lpPerformanceCount As Long) As Long
    Public Declare Function QueryPerformanceFrequency Lib "kernel32" (ByRef
lpFrequency As Long) As Long

    Private Declare Function timeBeginPeriod Lib "winmm.dll" (ByVal uPeriod As
UInt32) As UInt32
    Private Declare Function timeEndPeriod Lib "winmm.dll" (ByVal uPeriod As
UInt32) As UInt32
    Private Declare Function timeGetTime Lib "winmm.dll" () As UInt32

    Private uPeriod As UInt32 = CUInt(1)



    Sub Main()

        'Declarations
        Dim ComPortName As String = ""
        Dim LocalExit As Boolean = False
        Dim a As Long = 0
        Dim FileValid As Boolean = True
        Dim ThreadsLaunched As Boolean = False
        Dim LocalLaserControlMode As Long
        'Dim OutputLogFileACL As FileSecurity
        '

        'Initialization

        'set the system timer to update at 1 ms intervals (normally 15.6),
which is the max
        'the following timeBeginPeriod API call MUST have a corresponding
timeEndPeriod API call!!!!!!!!!!!
        timeBeginPeriod(uPeriod)
        'it will appear at the end of the program

        Application.EnableVisualStyles()
```

```
        SyncLock CentralClass.FileOpsLock
            'record the starting time
            If TimeZone.CurrentTimeZone.IsDaylightSavingTime(Now()) = True Then
                'daylight saving time
                CentralClass.ProgramStartTime = Format(Now(), "MMMM d, yyyy
HH:mm:ss ") & TimeZone.CurrentTimeZone.DaylightName
            Else
                'standard time
                CentralClass.ProgramStartTime = Format(Now(), "MMMM d, yyyy
HH:mm:ss ") & TimeZone.CurrentTimeZone.StandardName
            End If

        End SyncLock

        'set the process priority
        Process.GetCurrentProcess.PriorityClass =
ProcessPriorityClass.AboveNormal
        Thread.CurrentThread.Name = "Launcher"

        HelpBox.Hide()

        SyncLock Windowsill.RunBoxLock
            Windowsill.SpeedControlReady = False
            Windowsill.LaserPortReady = False
        End SyncLock


        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = False
            CentralClass.ExecutionStage = 0
        End SyncLock


        StartupForm.Panel1.BorderStyle = BorderStyle.Fixed3D
        For Each ComPortName In My.Computer.Ports.SerialPortNames
            'Adds the computer's serial ports to port list
            StartupForm.ControllerPortComboBox.Items.Add(ComPortName)
            StartupForm.LaserPortComboBox.Items.Add(ComPortName)
        Next

        Do
            StartupForm.ShowDialog()
            SyncLock CentralClass.ProgramExitLock
                LocalExit = CentralClass.ExitProgram
            End SyncLock
            If LocalExit = True Then
                'The User has chosen to Exit the application
                Beep()
                Exit Do
            Else
                'proceed with the program
                FileValid = True
                'check for valid filename
                SyncLock CentralClass.FileOpsLock
                    'check for valid name
                    Try
                        System.IO.Path.GetFullPath(CentralClass.Filename)
                    Catch ex As Exception
```

```
                        'invalid filename
                        FileValid = False
                        'alert the user
                        Beep()
                        MsgBox("The file " & CentralClass.Filename & " has
resulted in an exception:  " & vbNewLine & ex.Message)
                    End Try

                    'check if file, not directory, and for a good location
                    If FileValid = True Then
                        Try
                            'see if file exists; if it does, make sure it's not
a directory or read-only
                            If System.IO.File.Exists(CentralClass.Filename) =
True Then

                                'it exists; check for directory status
                                Try
                                    If
(System.IO.File.GetAttributes(CentralClass.Filename) And
System.IO.FileAttributes.Directory) = System.IO.FileAttributes.Directory Then
                                        'it's a directory
                                        FileValid = False
                                        Beep()
                                        MsgBox(CentralClass.Filename & " is a
directory.  Please choose something else.")
                                    End If
                                Catch ex1 As Exception
                                    'invalid filename or something
                                    FileValid = False
                                    'alert the user
                                    Beep()
                                    MsgBox("The file " & CentralClass.Filename
& " has resulted in an exception:  " & vbNewLine & ex1.Message)
                                End Try

                                'check for read-only
                                Try
                                    If
(System.IO.File.GetAttributes(CentralClass.Filename) And
System.IO.FileAttributes.ReadOnly) = System.IO.FileAttributes.ReadOnly Then
                                        'it's read-only
                                        FileValid = False
                                        Beep()
                                        MsgBox(CentralClass.Filename & " is a
read-only file.  Please choose something else.")
                                    End If
                                Catch ex1 As Exception
                                    'invalid filename or something
                                    FileValid = False
                                    'alert the user
                                    Beep()
                                    MsgBox("The file " & CentralClass.Filename
& " has resulted in an exception:  " & vbNewLine & ex1.Message)
                                End Try

                                'check for write and modify permissions
                                'maybe we'll just do this later by actually
opening the file
```

```
                                'Try
                                '    'get the ACL
                                '    OutputLogFileACL =
File.GetAccessControl(CentralClass.Filename, AccessControlSections.All)
                                '
'MsgBox(OutputLogFileACL.GetAccessRules(True, True,
GetType(System.Security.Principal.NTAccount)))
                                '    'If (OutputLogFileACL.GetAccessRules(True,
True, GetType(FileSystemRights)) And FileSystemRights.Write) <>
FileSystemRights.Write Then
                                '    'no write permission
                                '    FileValid = False
                                '    Beep()
                                '    Dim accessrules =
OutputLogFileACL.GetAccessRules(True, True,
GetType(System.Security.Principal.NTAccount))
                                '    accessrules()
                                '    For Each rule As
System.Security.AccessControl.FileSystemAccessRule In accessRules
                                '        MsgBox(rule.IdentityReference.Value)
                                '
MsgBox(rule.AccessControlType.ToString())
                                '
MsgBox(rule.FileSystemRights.ToString())
                                '    Next

                                '    MsgBox(CentralClass.Filename & " is not
writable by you.  Please choose something else.")
                                '    'End If
                                'Catch ex1 As Exception
                                '    'invalid filename or something
                                '    FileValid = False
                                '    'alert the user
                                '    Beep()
                                '    MsgBox("The file " & CentralClass.Filename
& " has resulted in an exception:  " & vbNewLine & ex1.Message)
                                'End Try

                                'see if the user REALLY wants to modify the
file
                                If FileValid = True Then
                                    Beep()
                                    If (CentralClass.AppendFile = True) Then
                                        'for append
                                        If MsgBox("The file " & vbNewLine &
CentralClass.Filename & vbNewLine & "already exists.  Do you really want to
append it?", MsgBoxStyle.YesNo, "Pin a tail on a donkey?") <> MsgBoxResult.Yes
Then
                                            'chose not to add on to it
                                            FileValid = False
                                        End If
                                    Else
                                        'for replace
                                        If MsgBox("The file " & vbNewLine &
CentralClass.Filename & vbNewLine & "already exists.  Do you really want to
replace it?  " & vbNewLine & vbNewLine & "THIS WILL ERASE ALL DATA CURRENTLY IN
THE FILE", MsgBoxStyle.YesNo, "Wash your brains?") <> MsgBoxResult.Yes Then
                                            'chose not to add on to it
                                            FileValid = False
```

```
                                    End If
                                End If
                            End If

                        Else
                            'the file does not already exist, see if the
directory does
                            Try
                                If
System.IO.Directory.Exists(System.IO.Path.GetDirectoryName(CentralClass.Filenam
e)) = False Then
                                    'path does not exist
                                    FileValid = False
                                    Beep()
                                    MsgBox(CentralClass.Filename & " does
not have a valid path.  Please choose something else.")
                                End If
                            Catch ex1 As Exception
                                'invalid filename or something
                                FileValid = False
                                'alert the user
                                Beep()
                                MsgBox("The file " & CentralClass.Filename
& " has resulted in an exception:  " & vbNewLine & ex1.Message)
                            End Try

                        End If

                    Catch ex As Exception
                        'invalid filename
                        FileValid = False
                        'alert the user
                        Beep()
                        MsgBox("The file " & CentralClass.Filename & " has
resulted in an exception:  " & vbNewLine & ex.Message)
                    End Try

                End If

                'prevent the creation of blank files due to no speed port
selection
                SyncLock CentralClass.FileOpsLock
                    If CentralClass.SpeedPort = "(none)" Then
                        FileValid = False
                        Exit Do
                    End If
                End SyncLock

                'Now see if we can open the file for editing
                If FileValid = True Then
                    SyncLock Lumberjack.LogFileLock
                        'try opening the file for writing

                        Try
                            Lumberjack.OutputLogFile =
System.IO.TextWriter.Synchronized(My.Computer.FileSystem.OpenTextFileWriter(Cen
tralClass.Filename, CentralClass.AppendFile))
                        Catch ex As Exception
                            'something did not work
```

```
                                FileValid = False
                                'alert the user
                                Beep()
                                MsgBox("The file " & CentralClass.Filename & "
has resulted in an exception:  " & vbNewLine & ex.Message)
                            End Try

                        End SyncLock

                    End If

                End SyncLock
            End If
        Loop While FileValid = False

        HelpBox.Hide()

        'just an end program box
        'TestEnder.Show()
        Application.DoEvents()

        Thread.CurrentThread.Priority = ThreadPriority.AboveNormal
        'Get ready to launch the other threads
        SyncLock CentralClass.ProgramExitLock
            LocalExit = CentralClass.ExitProgram
        End SyncLock

        'check for a selected speed controller port, and see if there is a
laser port
        If LocalExit = False Then
            SyncLock CentralClass.FileOpsLock
                'see if a port was chosen for the speed controller
                If CentralClass.SpeedPort = "(none)" Then
                    'no controller port, proceed to exit program
                    Beep()
                    SyncLock CentralClass.ProgramExitLock
                        CentralClass.ExitProgram = True
                        LocalExit = True
                    End SyncLock
                    MsgBox("You have not selected a serial port.  The Program
will not continue.", , "Uh huh")
                End If
                'see what the user selected for the laser port
                If CentralClass.LaserPort = "(none)" Then
                    'no port at all for laser
                    LocalLaserControlMode = 0
                Else
                    'A port was selected, see what it is
                    If CentralClass.LaserPort = CentralClass.SpeedPort Then
                        'The laser and speed controller share the same serial
port
                        LocalLaserControlMode = 1
                    Else
                        'The laser has its own independent serial port
                        LocalLaserControlMode = 2
                    End If
                End If
            End SyncLock
        End If
```

```
        SyncLock Longshoreman.LaserPortLock
            Longshoreman.LaserControlMode = LocalLaserControlMode
        End SyncLock

        If LocalExit = False Then
            'launch the threads, otherwise continue straight to shutdown

            'start the logger thread
            SyncLock CentralClass.ProgramExitLock
                'Program Status=Launching Threads
                CentralClass.ExecutionStage = 10
            End SyncLock
            Lumberjack.BostonLogger.Start()
            ThreadsLaunched = True
            Thread.Sleep(55)
            Application.DoEvents()
            'echo laser port mode to log
            If LocalLaserControlMode = 0 Then
                'No laser port
                Lumberjack.SendToLog("Starting Up in Speed Control Only Mode")
            ElseIf LocalLaserControlMode = 1 Then
                'laser port is shared with speed controller port
                Lumberjack.SendToLog("Starting Up in Shared Speed and Laser
Port Mode")
            ElseIf LocalLaserControlMode = 2 Then
                'laser port is separate from the speed controller port
                Lumberjack.SendToLog("Starting Up in Dual Port Mode")
            Else
                'what mode is this?
                Lumberjack.SendToLog("What Laser Control Mode is associated
with a value of " & Convert.ToString(LocalLaserControlMode) & "?")
            End If


            'start the user interface thread
            SyncLock CentralClass.ProgramExitLock
                'Program Status=Launching Threads
                CentralClass.ExecutionStage = 20
            End SyncLock
            Lumberjack.SendToLog("Starting the User Interface Module")
            Windowsill.Gooey.Start()
            Thread.Sleep(55)
            Application.DoEvents()

            'start the speed controller thread
            SyncLock CentralClass.ProgramExitLock
                'Program Status=Launching Threads
                CentralClass.ExecutionStage = 30
            End SyncLock
            Lumberjack.SendToLog("Starting the Controller Port Module")
            Stevedore.Col_Sanders.Priority = ThreadPriority.AboveNormal
            Stevedore.Col_Sanders.Start()
            Thread.Sleep(55)
            Application.DoEvents()

            'start the laser port thread, if necessary
            If LocalLaserControlMode = 2 Then
                'laser on a separate port
```

```
        SyncLock CentralClass.ProgramExitLock
            'Program Status=Launching Threads
            CentralClass.ExecutionStage = 40
        End SyncLock
        Lumberjack.SendToLog("Starting the Laser Port Module")
        Longshoreman.LaserPorter.Start()
        Thread.Sleep(55)
        Application.DoEvents()
    End If
    Thread.Sleep(55)
    Application.DoEvents()

End If

SyncLock CentralClass.ProgramExitLock
    LocalExit = CentralClass.ExitProgram
    If LocalExit = False Then
        CentralClass.ExecutionStage = 100
        'Program Status=Threads Launched, Running
    End If
End SyncLock
If LocalExit = False Then
    'Log the completion of startup procedures
    Lumberjack.SendToLog("All modules launched, entering Running Mode")
End If

'Now sleep until shutdown
Do
    'make sure threads are still alive
    'check logger
    If Lumberjack.BostonLogger.IsAlive = False Then
        'The Logger Died!  End program
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = True
        End SyncLock
    End If
    'check speed controller
    If Stevedore.Col_Sanders.IsAlive = False Then
        'The Speed Controller Died!  End program
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = True
        End SyncLock
    End If
    'check user interface
    If Windowsill.Gooey.IsAlive = False Then
        'The User Interface Died!  End program
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = True
        End SyncLock
    End If
    'check secondary serial port, if in that mode
    If LocalLaserControlMode = 2 Then
        If Longshoreman.LaserPorter.IsAlive = False Then
            'The Laser Port Died!  End program
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End If
    End If
```

```
            SyncLock CentralClass.ProgramExitLock
                LocalExit = CentralClass.ExitProgram
                If LocalExit = True Then
                    'Set Program Status to Beginning Shutdown
                    CentralClass.ExecutionStage = 1000
                End If
            End SyncLock
            Thread.Sleep(55)
            Application.DoEvents()
        Loop While LocalExit = False

        'shutdown stuff
        'make sure that the program status is Beginning Shutdown
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExecutionStage = 1000
        End SyncLock

        If ThreadsLaunched = True Then
            'threads were launched, log the shutdown
            Lumberjack.SendToLog("Program Shutdown in Progress")
        End If



        'shut down the speed controller
        Application.DoEvents()
        Thread.Sleep(55)
        If ThreadsLaunched = True Then
            'enter imminent controller shutdown into log
            Lumberjack.SendToLog("Initiate Speed Control Port Termination")
        End If
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExecutionStage = 1010
            'stage that tells the speed controller to finish
        End SyncLock
        Application.DoEvents()
        'now give the thread a few seconds to finish
        Thread.Sleep(55)
        Try
            If Stevedore.Col_Sanders.Join(60000) = False Then
                'it did not finish in time
                'it had a full minute (60000 ms)
                'kill it
                Stevedore.Col_Sanders.Abort()
                'log the issue
                If ThreadsLaunched = True Then
                    Lumberjack.SendToLog("The Speed Control Port Thread
Shutdown timed out.  The thread was aborted.")
                End If
            End If
        Catch ex As Exception
            'thread hadn't been started
        End Try
        Application.DoEvents()
        Thread.Sleep(55)


        'shut down the laser port
```

```
        Application.DoEvents()
        Thread.Sleep(55)
        If (ThreadsLaunched = True) And (LocalLaserControlMode = 2) Then
            'enter imminent laser port shutdown into log
            Lumberjack.SendToLog("Initiate Laser Port Termination")
        End If
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExecutionStage = 1020
            'stage that tells the laser port module to finish
        End SyncLock
        Application.DoEvents()
        Thread.Sleep(55)
        'now give the thread a few seconds to finish
        Try
            If Longshoreman.LaserPorter.Join(60000) = False Then
                'it did not finish in time
                'it had a full minute (60000 ms)
                'kill it
                Longshoreman.LaserPorter.Abort()
                'log the issue
                If ThreadsLaunched = True Then
                    Lumberjack.SendToLog("The Laser Port Thread Shutdown timed
out.  The thread was aborted.")
                End If
            End If
        Catch ex As Exception
            'thread hadn't been started
        End Try
        Application.DoEvents()
        Thread.Sleep(55)


        'shut down the user interface
        Application.DoEvents()
        Thread.Sleep(55)
        If ThreadsLaunched = True Then
            'enter imminent UI shutdown into log
            Lumberjack.SendToLog("Initiate User Interface Termination")
        End If
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExecutionStage = 1050
            'stage that tells the user interface to finish
        End SyncLock
        Application.DoEvents()
        Thread.Sleep(55)
        'now give the thread a few seconds to finish
        Try
            If Windowsill.Gooey.Join(5000) = False Then
                'it did not finish in time
                'kill it
                Windowsill.Gooey.Abort()
                'log the issue
                If ThreadsLaunched = True Then
                    Lumberjack.SendToLog("The User Interface Thread Shutdown
timed out.  The thread was aborted.")
                End If
            End If
        Catch ex As Exception
            'thread hadn't been started
```

```
        End Try
        Application.DoEvents()
        Thread.Sleep(100)


        'shut down the logger
        Application.DoEvents()
        Thread.Sleep(100)
        If ThreadsLaunched = True Then
            'enter imminent logger shutdown into log
            Lumberjack.SendToLog("Initiate Logger Termination")
        End If
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExecutionStage = 1100
            'stage that tells the logger to finish
        End SyncLock
        Application.DoEvents()
        Thread.Sleep(55)
        'now give the thread a few seconds to finish
        Try
            If Lumberjack.BostonLogger.Join(5000) = False Then
                'it did not finish in time
                'log the issue
                If ThreadsLaunched = True Then
                    Lumberjack.SendToLog("The Logger Thread Shutdown timed out.
The thread will be aborted.")
                End If
                Application.DoEvents()
                Thread.Sleep(250)
                Application.DoEvents()
                'kill it
                Lumberjack.BostonLogger.Abort()
            End If
        Catch ex As Exception
            'thread hadn't been started
        End Try
        Application.DoEvents()
        Thread.Sleep(55)



        Application.DoEvents()
        Thread.Sleep(100)

        'Close any resources that are still open (files, ports, forms)

        'Close and Dispose all forms

        'Dispose the FirePulse box
        Try
            FirePulse.Hide()
            FirePulse.Dispose()
        Catch ex As ObjectDisposedException
        End Try

        'Dispose the Helpbox
        Try
            HelpBox.Hide()
            HelpBox.Dispose()
```

```
        Catch ex As ObjectDisposedException
        End Try

        'Dispose the InsertComment box
        Try
            InsertComment.Hide()
            InsertComment.Dispose()
        Catch ex As ObjectDisposedException
        End Try

        'Dispose the Runbox
        Try
            RunBox.Hide()
            RunBox.Dispose()
        Catch ex As ObjectDisposedException
        End Try

        'Dispose the SendLaserCommand box
        Try
            SendLaserCommand.Hide()
            SendLaserCommand.Dispose()
        Catch ex As ObjectDisposedException
        End Try

        'Dispose the StartupForm
        Try
            StartupForm.Hide()
            StartupForm.Dispose()
        Catch ex As ObjectDisposedException
        End Try

        'Dispose the TestEnder box
        'Try
        '    TestEnder.Hide()
        '    TestEnder.Dispose()
        'Catch ex As ObjectDisposedException
        'End Try

        Application.DoEvents()

        'Force-Close the log file
        Try
            Lumberjack.OutputLogFile.Flush()
            Lumberjack.OutputLogFile.Close()
            Lumberjack.OutputLogFile.Dispose()
        Catch ex1 As ArgumentNullException
            'never initialized the output file
        Catch ex2 As ObjectDisposedException
            'already closed
        Catch ex3 As NullReferenceException
            'file never opened?
        Catch ex As Exception
            'something else
            MsgBox("An exception occurred.  " & vbNewLine & ex.Message, ,
"Argh!")
        End Try

        'Force-Close Serial Port 1 (Controller Port and possibly includes the
laser port)
```

```
        Try
            If Stevedore.COMPort1.IsOpen = True Then
                Stevedore.COMPort1.DiscardOutBuffer()
            End If
            Stevedore.COMPort1.Close()
            Stevedore.COMPort1.Dispose()
        Catch ex1 As InvalidOperationException
            'port is not open
        Catch ex As Exception
            'something else
            MsgBox("An exception occurred.  " & vbNewLine & ex.Message, ,
"Argh!")
        End Try

        'Force-Close Serial Port 2 (laser port, if any and on a separate port)
        Try
            If Longshoreman.COMPort2.IsOpen = True Then
                Longshoreman.COMPort2.DiscardOutBuffer()
            End If
            Longshoreman.COMPort2.Close()
            Longshoreman.COMPort2.Dispose()
        Catch ex1 As InvalidOperationException
            'port is not open
        Catch ex As Exception
            'something else
            MsgBox("An exception occurred.  " & vbNewLine & ex.Message, ,
"Argh!")
        End Try

        'THIS IS THE CORRESPONDING API CALL TO THE timeBeginPeriod CALL AT THE
BEGINNING OF THE PROGRAM
        timeEndPeriod(uPeriod)

        Application.DoEvents()
        Thread.Sleep(1000)

        'Then leave the program
        Application.Exit()

    End Sub



End Module



Friend NotInheritable Class CentralClass

    'Application-wide variables
    'The following blocks each have their own undefined new object
    'That object must be used for synclocking that particular block
    'Otherwise, synchronization issues will readily occur
    '
    '
    'File and Startup variables
    Friend Shared Filename As String = ""
    Friend Shared AppendFile As Boolean = True
    Friend Shared AdvancedTiming As Boolean = False
```

```
    Friend Shared SpeedPort As String = ""
    Friend Shared LaserPort As String = ""
    Friend Shared ComPortBaudRate As Integer = 9600
    Friend Shared ComPortParity As System.IO.Ports.Parity =
System.IO.Ports.Parity.None
    Friend Shared ComPortFlowControl As System.IO.Ports.Handshake =
System.IO.Ports.Handshake.None
    Friend Shared ComPortStopBits As System.IO.Ports.StopBits =
System.IO.Ports.StopBits.One
    Friend Shared ComPortDataBits As Integer = 8
    Friend Shared ComPortEncoding As System.Text.Encoding =
System.Text.Encoding.ASCII
    Friend Shared LaserNewlineString As String = vbCr
    Friend Shared ProgramStartTime As String = ""
    Friend Shared FileOpsLock As New Object
    '
    '
    'Program Control/Exit
    Friend Shared ExecutionStage As Long = 0
    Friend Shared ExitProgram As Boolean = False
    Friend Shared ProgramExitLock As New Object
    '
    '



End Class
```

B.11 RUNBOX.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class RunBox
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(RunBox))
        Me.TimeBox = New System.Windows.Forms.Label
```

```
Me.DateBox = New System.Windows.Forms.Label
Me.Label3 = New System.Windows.Forms.Label
Me.Label4 = New System.Windows.Forms.Label
Me.Label5 = New System.Windows.Forms.Label
Me.CurrentSpeedBox = New System.Windows.Forms.Label
Me.Label7 = New System.Windows.Forms.Label
Me.OutputFileBox = New System.Windows.Forms.Label
Me.Label9 = New System.Windows.Forms.Label
Me.CavitationBox = New System.Windows.Forms.Label
Me.Label11 = New System.Windows.Forms.Label
Me.SpeedInRangeBox = New System.Windows.Forms.Label
Me.Label13 = New System.Windows.Forms.Label
Me.CurrentStatusBox = New System.Windows.Forms.Label
Me.GroupBox1 = New System.Windows.Forms.GroupBox
Me.Label18 = New System.Windows.Forms.Label
Me.RestartDelayShowBox = New System.Windows.Forms.Label
Me.RestartDelayInputBox = New System.Windows.Forms.TextBox
Me.Label17 = New System.Windows.Forms.Label
Me.SetSpeedShowBox = New System.Windows.Forms.Label
Me.SetSpeedInputBox = New System.Windows.Forms.TextBox
Me.Label15 = New System.Windows.Forms.Label
Me.ButtonSetSpeed = New System.Windows.Forms.Button
Me.OnCavitationStopAndRestart = New System.Windows.Forms.RadioButton
Me.OnCavitationStop = New System.Windows.Forms.RadioButton
Me.ButtonStopSpeed = New System.Windows.Forms.Button
Me.ButtonStartSpeed = New System.Windows.Forms.Button
Me.OnCavitationContinue = New System.Windows.Forms.RadioButton
Me.GroupBox2 = New System.Windows.Forms.GroupBox
Me.Label28 = New System.Windows.Forms.Label
Me.NextPulsePowerShowBox = New System.Windows.Forms.Label
Me.ButtonSetLaser = New System.Windows.Forms.Button
Me.Label26 = New System.Windows.Forms.Label
Me.EstimatedDurationShowBox = New System.Windows.Forms.Label
Me.EstimatedDurationInputBox = New System.Windows.Forms.TextBox
Me.Label23 = New System.Windows.Forms.Label
Me.LaserDelayShowBox = New System.Windows.Forms.Label
Me.LaserDelayInputBox = New System.Windows.Forms.TextBox
Me.FireLaserBox = New System.Windows.Forms.Label
Me.ButtonFireLaser = New System.Windows.Forms.Button
Me.ButtonSendLaserCommand = New System.Windows.Forms.Button
Me.Label22 = New System.Windows.Forms.Label
Me.AutomaticPulsingFalse = New System.Windows.Forms.RadioButton
Me.Label21 = New System.Windows.Forms.Label
Me.LaserRxBox = New System.Windows.Forms.TextBox
Me.AutomaticPulsingTrue = New System.Windows.Forms.RadioButton
Me.ButtonInsertComment = New System.Windows.Forms.Button
Me.ButtonEnd = New System.Windows.Forms.Button
Me.ButtonHelp = New System.Windows.Forms.Button
Me.LogTrace = New System.Windows.Forms.TextBox
Me.Label20 = New System.Windows.Forms.Label
Me.GroupBox1.SuspendLayout()
Me.GroupBox2.SuspendLayout()
Me.SuspendLayout()
'
'TimeBox
'
Me.TimeBox.AutoSize = True
Me.TimeBox.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
Me.TimeBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
```

```
        Me.TimeBox.Location = New System.Drawing.Point(96, 9)
        Me.TimeBox.MaximumSize = New System.Drawing.Size(134, 15)
        Me.TimeBox.MinimumSize = New System.Drawing.Size(134, 15)
        Me.TimeBox.Name = "TimeBox"
        Me.TimeBox.Size = New System.Drawing.Size(134, 15)
        Me.TimeBox.TabIndex = 1
        Me.TimeBox.Text = "TIME"
        '
        'DateBox
        '
        Me.DateBox.AutoSize = True
        Me.DateBox.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.DateBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.DateBox.Location = New System.Drawing.Point(314, 9)
        Me.DateBox.MaximumSize = New System.Drawing.Size(296, 15)
        Me.DateBox.MinimumSize = New System.Drawing.Size(296, 15)
        Me.DateBox.Name = "DateBox"
        Me.DateBox.Size = New System.Drawing.Size(296, 15)
        Me.DateBox.TabIndex = 3
        Me.DateBox.Text = "DATE"
        '
        'Label3
        '
        Me.Label3.AutoSize = True
        Me.Label3.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label3.Location = New System.Drawing.Point(57, 9)
        Me.Label3.MinimumSize = New System.Drawing.Size(33, 13)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(33, 13)
        Me.Label3.TabIndex = 0
        Me.Label3.Text = "Time:"
        '
        'Label4
        '
        Me.Label4.AutoSize = True
        Me.Label4.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label4.Location = New System.Drawing.Point(275, 9)
        Me.Label4.MinimumSize = New System.Drawing.Size(33, 13)
        Me.Label4.Name = "Label4"
        Me.Label4.Size = New System.Drawing.Size(33, 13)
        Me.Label4.TabIndex = 2
        Me.Label4.Text = "Date:"
        '
        'Label5
        '
        Me.Label5.AutoSize = True
        Me.Label5.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label5.Location = New System.Drawing.Point(12, 24)
        Me.Label5.MinimumSize = New System.Drawing.Size(78, 13)
        Me.Label5.Name = "Label5"
        Me.Label5.Size = New System.Drawing.Size(78, 13)
        Me.Label5.TabIndex = 4
        Me.Label5.Text = "Current Speed:"
        '
        'CurrentSpeedBox
        '
        Me.CurrentSpeedBox.AutoSize = True
        Me.CurrentSpeedBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
```

```
        Me.CurrentSpeedBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.CurrentSpeedBox.Location = New System.Drawing.Point(96, 24)
        Me.CurrentSpeedBox.MaximumSize = New System.Drawing.Size(134, 15)
        Me.CurrentSpeedBox.MinimumSize = New System.Drawing.Size(134, 15)
        Me.CurrentSpeedBox.Name = "CurrentSpeedBox"
        Me.CurrentSpeedBox.Size = New System.Drawing.Size(134, 15)
        Me.CurrentSpeedBox.TabIndex = 5
        Me.CurrentSpeedBox.Text = "SPEED"
        '
        'Label7
        '
        Me.Label7.AutoSize = True
        Me.Label7.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label7.Location = New System.Drawing.Point(247, 24)
        Me.Label7.MinimumSize = New System.Drawing.Size(61, 13)
        Me.Label7.Name = "Label7"
        Me.Label7.Size = New System.Drawing.Size(61, 13)
        Me.Label7.TabIndex = 6
        Me.Label7.Text = "Output File:"
        '
        'OutputFileBox
        '
        Me.OutputFileBox.AutoEllipsis = True
        Me.OutputFileBox.AutoSize = True
        Me.OutputFileBox.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.OutputFileBox.Cursor = System.Windows.Forms.Cursors.Default
        Me.OutputFileBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.OutputFileBox.Location = New System.Drawing.Point(314, 24)
        Me.OutputFileBox.MaximumSize = New System.Drawing.Size(296, 15)
        Me.OutputFileBox.MinimumSize = New System.Drawing.Size(296, 15)
        Me.OutputFileBox.Name = "OutputFileBox"
        Me.OutputFileBox.Size = New System.Drawing.Size(296, 15)
        Me.OutputFileBox.TabIndex = 7
        Me.OutputFileBox.Text = "FILE"
        '
        'Label9
        '
        Me.Label9.AutoSize = True
        Me.Label9.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label9.Location = New System.Drawing.Point(391, 54)
        Me.Label9.MinimumSize = New System.Drawing.Size(104, 13)
        Me.Label9.Name = "Label9"
        Me.Label9.Size = New System.Drawing.Size(104, 13)
        Me.Label9.TabIndex = 12
        Me.Label9.Text = "Cavitation Detected:"
        '
        'CavitationBox
        '
        Me.CavitationBox.AutoSize = True
        Me.CavitationBox.BackColor = System.Drawing.Color.Navy
        Me.CavitationBox.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.CavitationBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.CavitationBox.Font = New System.Drawing.Font("Microsoft Sans Serif",
8.25!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
        Me.CavitationBox.ForeColor = System.Drawing.Color.Yellow
        Me.CavitationBox.Location = New System.Drawing.Point(501, 54)
        Me.CavitationBox.MaximumSize = New System.Drawing.Size(40, 15)
```

```
        Me.CavitationBox.MinimumSize = New System.Drawing.Size(40, 15)
        Me.CavitationBox.Name = "CavitationBox"
        Me.CavitationBox.Size = New System.Drawing.Size(40, 15)
        Me.CavitationBox.TabIndex = 13
        Me.CavitationBox.Text = "NO"
        '
        'Label11
        '
        Me.Label11.AutoSize = True
        Me.Label11.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label11.Location = New System.Drawing.Point(97, 54)
        Me.Label11.MinimumSize = New System.Drawing.Size(87, 13)
        Me.Label11.Name = "Label11"
        Me.Label11.Size = New System.Drawing.Size(87, 13)
        Me.Label11.TabIndex = 10
        Me.Label11.Text = "Speed in Range:"
        '
        'SpeedInRangeBox
        '
        Me.SpeedInRangeBox.AutoSize = True
        Me.SpeedInRangeBox.BackColor = System.Drawing.Color.Navy
        Me.SpeedInRangeBox.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.SpeedInRangeBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.SpeedInRangeBox.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.SpeedInRangeBox.ForeColor = System.Drawing.Color.Yellow
        Me.SpeedInRangeBox.Location = New System.Drawing.Point(190, 54)
        Me.SpeedInRangeBox.MaximumSize = New System.Drawing.Size(40, 15)
        Me.SpeedInRangeBox.MinimumSize = New System.Drawing.Size(40, 15)
        Me.SpeedInRangeBox.Name = "SpeedInRangeBox"
        Me.SpeedInRangeBox.Size = New System.Drawing.Size(40, 15)
        Me.SpeedInRangeBox.TabIndex = 11
        Me.SpeedInRangeBox.Text = "NO"
        '
        'Label13
        '
        Me.Label13.AutoSize = True
        Me.Label13.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label13.Location = New System.Drawing.Point(50, 41)
        Me.Label13.MinimumSize = New System.Drawing.Size(40, 13)
        Me.Label13.Name = "Label13"
        Me.Label13.Size = New System.Drawing.Size(40, 13)
        Me.Label13.TabIndex = 8
        Me.Label13.Text = "Status:"
        '
        'CurrentStatusBox
        '
        Me.CurrentStatusBox.AutoSize = True
        Me.CurrentStatusBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.CurrentStatusBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.CurrentStatusBox.Location = New System.Drawing.Point(96, 39)
        Me.CurrentStatusBox.MaximumSize = New System.Drawing.Size(514, 15)
        Me.CurrentStatusBox.MinimumSize = New System.Drawing.Size(514, 15)
        Me.CurrentStatusBox.Name = "CurrentStatusBox"
        Me.CurrentStatusBox.Size = New System.Drawing.Size(514, 15)
        Me.CurrentStatusBox.TabIndex = 9
```

```
        Me.CurrentStatusBox.Text = "STATUS"
        '
        'GroupBox1
        '
        Me.GroupBox1.Controls.Add(Me.Label18)
        Me.GroupBox1.Controls.Add(Me.RestartDelayShowBox)
        Me.GroupBox1.Controls.Add(Me.RestartDelayInputBox)
        Me.GroupBox1.Controls.Add(Me.Label17)
        Me.GroupBox1.Controls.Add(Me.SetSpeedShowBox)
        Me.GroupBox1.Controls.Add(Me.SetSpeedInputBox)
        Me.GroupBox1.Controls.Add(Me.Label15)
        Me.GroupBox1.Controls.Add(Me.ButtonSetSpeed)
        Me.GroupBox1.Controls.Add(Me.OnCavitationStopAndRestart)
        Me.GroupBox1.Controls.Add(Me.OnCavitationStop)
        Me.GroupBox1.Controls.Add(Me.ButtonStopSpeed)
        Me.GroupBox1.Controls.Add(Me.ButtonStartSpeed)
        Me.GroupBox1.Controls.Add(Me.OnCavitationContinue)
        Me.GroupBox1.Location = New System.Drawing.Point(12, 72)
        Me.GroupBox1.MaximumSize = New System.Drawing.Size(296, 147)
        Me.GroupBox1.MinimumSize = New System.Drawing.Size(296, 147)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(296, 147)
        Me.GroupBox1.TabIndex = 14
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "Speed Controls"
        '
        'Label18
        '
        Me.Label18.AutoSize = True
        Me.Label18.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label18.Location = New System.Drawing.Point(64, 91)
        Me.Label18.MaximumSize = New System.Drawing.Size(74, 13)
        Me.Label18.MinimumSize = New System.Drawing.Size(74, 13)
        Me.Label18.Name = "Label18"
        Me.Label18.Size = New System.Drawing.Size(74, 13)
        Me.Label18.TabIndex = 7
        Me.Label18.Text = "Restart Delay:"
        '
        'RestartDelayShowBox
        '
        Me.RestartDelayShowBox.AutoSize = True
        Me.RestartDelayShowBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.RestartDelayShowBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.RestartDelayShowBox.Location = New System.Drawing.Point(215, 91)
        Me.RestartDelayShowBox.MaximumSize = New System.Drawing.Size(65, 15)
        Me.RestartDelayShowBox.MinimumSize = New System.Drawing.Size(65, 15)
        Me.RestartDelayShowBox.Name = "RestartDelayShowBox"
        Me.RestartDelayShowBox.Size = New System.Drawing.Size(65, 15)
        Me.RestartDelayShowBox.TabIndex = 9
        Me.RestartDelayShowBox.Text = "00000"
        '
        'RestartDelayInputBox
        '
        Me.RestartDelayInputBox.Location = New System.Drawing.Point(144, 88)
        Me.RestartDelayInputBox.MaximumSize = New System.Drawing.Size(65, 20)
        Me.RestartDelayInputBox.MaxLength = 9
        Me.RestartDelayInputBox.MinimumSize = New System.Drawing.Size(65, 20)
        Me.RestartDelayInputBox.Name = "RestartDelayInputBox"
```

```
        Me.RestartDelayInputBox.Size = New System.Drawing.Size(65, 20)
        Me.RestartDelayInputBox.TabIndex = 8
        Me.RestartDelayInputBox.Text = "00000"
        '
        'Label17
        '
        Me.Label17.AutoSize = True
        Me.Label17.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label17.Location = New System.Drawing.Point(13, 21)
        Me.Label17.MaximumSize = New System.Drawing.Size(60, 13)
        Me.Label17.MinimumSize = New System.Drawing.Size(60, 13)
        Me.Label17.Name = "Label17"
        Me.Label17.Size = New System.Drawing.Size(60, 13)
        Me.Label17.TabIndex = 0
        Me.Label17.Text = "Set Speed:"
        '
        'SetSpeedShowBox
        '
        Me.SetSpeedShowBox.AutoSize = True
        Me.SetSpeedShowBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.SetSpeedShowBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.SetSpeedShowBox.Location = New System.Drawing.Point(16, 67)
        Me.SetSpeedShowBox.MaximumSize = New System.Drawing.Size(65, 15)
        Me.SetSpeedShowBox.MinimumSize = New System.Drawing.Size(65, 15)
        Me.SetSpeedShowBox.Name = "SetSpeedShowBox"
        Me.SetSpeedShowBox.Size = New System.Drawing.Size(65, 15)
        Me.SetSpeedShowBox.TabIndex = 2
        Me.SetSpeedShowBox.Text = "123456789"
        '
        'SetSpeedInputBox
        '
        Me.SetSpeedInputBox.Location = New System.Drawing.Point(16, 41)
        Me.SetSpeedInputBox.MaximumSize = New System.Drawing.Size(65, 20)
        Me.SetSpeedInputBox.MaxLength = 9
        Me.SetSpeedInputBox.MinimumSize = New System.Drawing.Size(65, 20)
        Me.SetSpeedInputBox.Name = "SetSpeedInputBox"
        Me.SetSpeedInputBox.Size = New System.Drawing.Size(65, 20)
        Me.SetSpeedInputBox.TabIndex = 1
        Me.SetSpeedInputBox.Text = "00000"
        '
        'Label15
        '
        Me.Label15.AutoSize = True
        Me.Label15.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label15.Location = New System.Drawing.Point(89, 21)
        Me.Label15.MaximumSize = New System.Drawing.Size(74, 13)
        Me.Label15.MinimumSize = New System.Drawing.Size(74, 13)
        Me.Label15.Name = "Label15"
        Me.Label15.Size = New System.Drawing.Size(74, 13)
        Me.Label15.TabIndex = 3
        Me.Label15.Text = "On Cavitation:"
        '
        'ButtonSetSpeed
        '
        Me.ButtonSetSpeed.Location = New System.Drawing.Point(6, 118)
        Me.ButtonSetSpeed.MaximumSize = New System.Drawing.Size(75, 23)
        Me.ButtonSetSpeed.MinimumSize = New System.Drawing.Size(75, 23)
        Me.ButtonSetSpeed.Name = "ButtonSetSpeed"
```

```
        Me.ButtonSetSpeed.Size = New System.Drawing.Size(75, 23)
        Me.ButtonSetSpeed.TabIndex = 10
        Me.ButtonSetSpeed.Text = "Set"
        Me.ButtonSetSpeed.UseVisualStyleBackColor = True
        '
        'OnCavitationStopAndRestart
        '
        Me.OnCavitationStopAndRestart.AutoSize = True
        Me.OnCavitationStopAndRestart.Checked = True
        Me.OnCavitationStopAndRestart.Location = New System.Drawing.Point(169,
65)
        Me.OnCavitationStopAndRestart.MaximumSize = New
System.Drawing.Size(115, 17)
        Me.OnCavitationStopAndRestart.MinimumSize = New
System.Drawing.Size(115, 17)
        Me.OnCavitationStopAndRestart.Name = "OnCavitationStopAndRestart"
        Me.OnCavitationStopAndRestart.Size = New System.Drawing.Size(115, 17)
        Me.OnCavitationStopAndRestart.TabIndex = 6
        Me.OnCavitationStopAndRestart.TabStop = True
        Me.OnCavitationStopAndRestart.Text = "Stop + Restart"
        Me.OnCavitationStopAndRestart.UseVisualStyleBackColor = True
        '
        'OnCavitationStop
        '
        Me.OnCavitationStop.AutoSize = True
        Me.OnCavitationStop.Location = New System.Drawing.Point(169, 42)
        Me.OnCavitationStop.MaximumSize = New System.Drawing.Size(115, 17)
        Me.OnCavitationStop.MinimumSize = New System.Drawing.Size(115, 17)
        Me.OnCavitationStop.Name = "OnCavitationStop"
        Me.OnCavitationStop.Size = New System.Drawing.Size(115, 17)
        Me.OnCavitationStop.TabIndex = 5
        Me.OnCavitationStop.Text = "Stop"
        Me.OnCavitationStop.UseVisualStyleBackColor = True
        '
        'ButtonStopSpeed
        '
        Me.ButtonStopSpeed.Enabled = False
        Me.ButtonStopSpeed.Location = New System.Drawing.Point(215, 118)
        Me.ButtonStopSpeed.MaximumSize = New System.Drawing.Size(75, 23)
        Me.ButtonStopSpeed.MinimumSize = New System.Drawing.Size(75, 23)
        Me.ButtonStopSpeed.Name = "ButtonStopSpeed"
        Me.ButtonStopSpeed.Size = New System.Drawing.Size(75, 23)
        Me.ButtonStopSpeed.TabIndex = 12
        Me.ButtonStopSpeed.Text = "Stop"
        Me.ButtonStopSpeed.UseVisualStyleBackColor = True
        '
        'ButtonStartSpeed
        '
        Me.ButtonStartSpeed.Location = New System.Drawing.Point(134, 118)
        Me.ButtonStartSpeed.MaximumSize = New System.Drawing.Size(75, 23)
        Me.ButtonStartSpeed.MinimumSize = New System.Drawing.Size(75, 23)
        Me.ButtonStartSpeed.Name = "ButtonStartSpeed"
        Me.ButtonStartSpeed.Size = New System.Drawing.Size(75, 23)
        Me.ButtonStartSpeed.TabIndex = 11
        Me.ButtonStartSpeed.Text = "Start"
        Me.ButtonStartSpeed.UseVisualStyleBackColor = True
        '
        'OnCavitationContinue
        '
```

```
        Me.OnCavitationContinue.AutoSize = True
        Me.OnCavitationContinue.Location = New System.Drawing.Point(169, 19)
        Me.OnCavitationContinue.MaximumSize = New System.Drawing.Size(115, 17)
        Me.OnCavitationContinue.MinimumSize = New System.Drawing.Size(115, 17)
        Me.OnCavitationContinue.Name = "OnCavitationContinue"
        Me.OnCavitationContinue.Size = New System.Drawing.Size(115, 17)
        Me.OnCavitationContinue.TabIndex = 4
        Me.OnCavitationContinue.Text = "Continue"
        Me.OnCavitationContinue.UseVisualStyleBackColor = True
        '
        'GroupBox2
        '
        Me.GroupBox2.Controls.Add(Me.Label28)
        Me.GroupBox2.Controls.Add(Me.NextPulsePowerShowBox)
        Me.GroupBox2.Controls.Add(Me.ButtonSetLaser)
        Me.GroupBox2.Controls.Add(Me.Label26)
        Me.GroupBox2.Controls.Add(Me.EstimatedDurationShowBox)
        Me.GroupBox2.Controls.Add(Me.EstimatedDurationInputBox)
        Me.GroupBox2.Controls.Add(Me.Label23)
        Me.GroupBox2.Controls.Add(Me.LaserDelayShowBox)
        Me.GroupBox2.Controls.Add(Me.LaserDelayInputBox)
        Me.GroupBox2.Controls.Add(Me.FireLaserBox)
        Me.GroupBox2.Controls.Add(Me.ButtonFireLaser)
        Me.GroupBox2.Controls.Add(Me.ButtonSendLaserCommand)
        Me.GroupBox2.Controls.Add(Me.Label22)
        Me.GroupBox2.Controls.Add(Me.AutomaticPulsingFalse)
        Me.GroupBox2.Controls.Add(Me.Label21)
        Me.GroupBox2.Controls.Add(Me.LaserRxBox)
        Me.GroupBox2.Controls.Add(Me.AutomaticPulsingTrue)
        Me.GroupBox2.Location = New System.Drawing.Point(314, 72)
        Me.GroupBox2.MaximumSize = New System.Drawing.Size(296, 293)
        Me.GroupBox2.MinimumSize = New System.Drawing.Size(296, 293)
        Me.GroupBox2.Name = "GroupBox2"
        Me.GroupBox2.Size = New System.Drawing.Size(296, 293)
        Me.GroupBox2.TabIndex = 15
        Me.GroupBox2.TabStop = False
        Me.GroupBox2.Text = "Laser Controls"
        '
        'Label28
        '
        Me.Label28.AutoSize = True
        Me.Label28.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label28.Location = New System.Drawing.Point(73, 44)
        Me.Label28.MaximumSize = New System.Drawing.Size(40, 39)
        Me.Label28.MinimumSize = New System.Drawing.Size(40, 39)
        Me.Label28.Name = "Label28"
        Me.Label28.Size = New System.Drawing.Size(40, 39)
        Me.Label28.TabIndex = 4
        Me.Label28.Text = "Next Pulse Power:"
        '
        'NextPulsePowerShowBox
        '
        Me.NextPulsePowerShowBox.AutoSize = True
        Me.NextPulsePowerShowBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.NextPulsePowerShowBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat
        Me.NextPulsePowerShowBox.Location = New System.Drawing.Point(69, 91)
        Me.NextPulsePowerShowBox.MaximumSize = New System.Drawing.Size(65, 15)
```

```
        Me.NextPulsePowerShowBox.MinimumSize = New System.Drawing.Size(65, 15)
        Me.NextPulsePowerShowBox.Name = "NextPulsePowerShowBox"
        Me.NextPulsePowerShowBox.Size = New System.Drawing.Size(65, 15)
        Me.NextPulsePowerShowBox.TabIndex = 5
        Me.NextPulsePowerShowBox.Text = "000001100"
        '
        'ButtonSetLaser
        '
        Me.ButtonSetLaser.Location = New System.Drawing.Point(6, 86)
        Me.ButtonSetLaser.MaximumSize = New System.Drawing.Size(57, 23)
        Me.ButtonSetLaser.MinimumSize = New System.Drawing.Size(57, 23)
        Me.ButtonSetLaser.Name = "ButtonSetLaser"
        Me.ButtonSetLaser.Size = New System.Drawing.Size(57, 23)
        Me.ButtonSetLaser.TabIndex = 12
        Me.ButtonSetLaser.Text = "Set"
        Me.ButtonSetLaser.UseVisualStyleBackColor = True
        '
        'Label26
        '
        Me.Label26.AutoSize = True
        Me.Label26.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label26.Location = New System.Drawing.Point(137, 37)
        Me.Label26.MaximumSize = New System.Drawing.Size(56, 26)
        Me.Label26.MinimumSize = New System.Drawing.Size(56, 26)
        Me.Label26.Name = "Label26"
        Me.Label26.Size = New System.Drawing.Size(56, 26)
        Me.Label26.TabIndex = 6
        Me.Label26.Text = "Estimated Duration:"
        '
        'EstimatedDurationShowBox
        '
        Me.EstimatedDurationShowBox.AutoSize = True
        Me.EstimatedDurationShowBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.EstimatedDurationShowBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat
        Me.EstimatedDurationShowBox.Location = New System.Drawing.Point(140,
91)
        Me.EstimatedDurationShowBox.MaximumSize = New System.Drawing.Size(65,
15)
        Me.EstimatedDurationShowBox.MinimumSize = New System.Drawing.Size(65,
15)
        Me.EstimatedDurationShowBox.Name = "EstimatedDurationShowBox"
        Me.EstimatedDurationShowBox.Size = New System.Drawing.Size(65, 15)
        Me.EstimatedDurationShowBox.TabIndex = 8
        Me.EstimatedDurationShowBox.Text = "00000"
        '
        'EstimatedDurationInputBox
        '
        Me.EstimatedDurationInputBox.Location = New System.Drawing.Point(140,
65)
        Me.EstimatedDurationInputBox.MaximumSize = New System.Drawing.Size(65,
20)
        Me.EstimatedDurationInputBox.MaxLength = 9
        Me.EstimatedDurationInputBox.MinimumSize = New System.Drawing.Size(65,
20)
        Me.EstimatedDurationInputBox.Name = "EstimatedDurationInputBox"
        Me.EstimatedDurationInputBox.Size = New System.Drawing.Size(65, 20)
        Me.EstimatedDurationInputBox.TabIndex = 7
```

```
        Me.EstimatedDurationInputBox.Text = "00000"
        '
        'Label23
        '
        Me.Label23.AutoSize = True
        Me.Label23.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label23.Location = New System.Drawing.Point(218, 44)
        Me.Label23.MaximumSize = New System.Drawing.Size(37, 13)
        Me.Label23.MinimumSize = New System.Drawing.Size(37, 13)
        Me.Label23.Name = "Label23"
        Me.Label23.Size = New System.Drawing.Size(37, 13)
        Me.Label23.TabIndex = 9
        Me.Label23.Text = "Delay:"
        '
        'LaserDelayShowBox
        '
        Me.LaserDelayShowBox.AutoSize = True
        Me.LaserDelayShowBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.LaserDelayShowBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.LaserDelayShowBox.Location = New System.Drawing.Point(220, 91)
        Me.LaserDelayShowBox.MaximumSize = New System.Drawing.Size(65, 15)
        Me.LaserDelayShowBox.MinimumSize = New System.Drawing.Size(65, 15)
        Me.LaserDelayShowBox.Name = "LaserDelayShowBox"
        Me.LaserDelayShowBox.Size = New System.Drawing.Size(65, 15)
        Me.LaserDelayShowBox.TabIndex = 11
        Me.LaserDelayShowBox.Text = "00000"
        '
        'LaserDelayInputBox
        '
        Me.LaserDelayInputBox.Location = New System.Drawing.Point(220, 65)
        Me.LaserDelayInputBox.MaximumSize = New System.Drawing.Size(65, 20)
        Me.LaserDelayInputBox.MaxLength = 9
        Me.LaserDelayInputBox.MinimumSize = New System.Drawing.Size(65, 20)
        Me.LaserDelayInputBox.Name = "LaserDelayInputBox"
        Me.LaserDelayInputBox.Size = New System.Drawing.Size(65, 20)
        Me.LaserDelayInputBox.TabIndex = 10
        Me.LaserDelayInputBox.Text = "00000"
        '
        'FireLaserBox
        '
        Me.FireLaserBox.AutoSize = True
        Me.FireLaserBox.BackColor = System.Drawing.Color.Crimson
        Me.FireLaserBox.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.FireLaserBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.FireLaserBox.Font = New System.Drawing.Font("Microsoft Sans Serif",
8.25!, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
        Me.FireLaserBox.ForeColor = System.Drawing.Color.Yellow
        Me.FireLaserBox.Location = New System.Drawing.Point(6, 44)
        Me.FireLaserBox.MaximumSize = New System.Drawing.Size(60, 30)
        Me.FireLaserBox.MinimumSize = New System.Drawing.Size(60, 30)
        Me.FireLaserBox.Name = "FireLaserBox"
        Me.FireLaserBox.Size = New System.Drawing.Size(60, 30)
        Me.FireLaserBox.TabIndex = 3
        Me.FireLaserBox.Text = "FIRE LASER!"
        Me.FireLaserBox.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
```

```
'ButtonFireLaser
'
Me.ButtonFireLaser.Location = New System.Drawing.Point(6, 118)
Me.ButtonFireLaser.Name = "ButtonFireLaser"
Me.ButtonFireLaser.Size = New System.Drawing.Size(128, 23)
Me.ButtonFireLaser.TabIndex = 13
Me.ButtonFireLaser.Text = "Fire Pulse..."
Me.ButtonFireLaser.UseVisualStyleBackColor = True
'
'ButtonSendLaserCommand
'
Me.ButtonSendLaserCommand.Location = New System.Drawing.Point(140, 118)
Me.ButtonSendLaserCommand.Name = "ButtonSendLaserCommand"
Me.ButtonSendLaserCommand.Size = New System.Drawing.Size(150, 23)
Me.ButtonSendLaserCommand.TabIndex = 14
Me.ButtonSendLaserCommand.Text = "Send Command to Laser"
Me.ButtonSendLaserCommand.UseVisualStyleBackColor = True
'
'Label22
'
Me.Label22.AutoSize = True
Me.Label22.FlatStyle = System.Windows.Forms.FlatStyle.Flat
Me.Label22.Location = New System.Drawing.Point(6, 21)
Me.Label22.MaximumSize = New System.Drawing.Size(110, 13)
Me.Label22.MinimumSize = New System.Drawing.Size(110, 13)
Me.Label22.Name = "Label22"
Me.Label22.Size = New System.Drawing.Size(110, 13)
Me.Label22.TabIndex = 0
Me.Label22.Text = "Automatic Pulsing:"
'
'AutomaticPulsingFalse
'
Me.AutomaticPulsingFalse.AutoSize = True
Me.AutomaticPulsingFalse.Checked = True
Me.AutomaticPulsingFalse.Location = New System.Drawing.Point(221, 19)
Me.AutomaticPulsingFalse.MaximumSize = New System.Drawing.Size(39, 17)
Me.AutomaticPulsingFalse.MinimumSize = New System.Drawing.Size(39, 17)
Me.AutomaticPulsingFalse.Name = "AutomaticPulsingFalse"
Me.AutomaticPulsingFalse.Size = New System.Drawing.Size(39, 17)
Me.AutomaticPulsingFalse.TabIndex = 2
Me.AutomaticPulsingFalse.TabStop = True
Me.AutomaticPulsingFalse.Text = "No"
Me.AutomaticPulsingFalse.UseVisualStyleBackColor = True
'
'Label21
'
Me.Label21.AutoSize = True
Me.Label21.FlatStyle = System.Windows.Forms.FlatStyle.Flat
Me.Label21.Location = New System.Drawing.Point(6, 150)
Me.Label21.MaximumSize = New System.Drawing.Size(128, 13)
Me.Label21.MinimumSize = New System.Drawing.Size(128, 13)
Me.Label21.Name = "Label21"
Me.Label21.Size = New System.Drawing.Size(128, 13)
Me.Label21.TabIndex = 15
Me.Label21.Text = "Received from Laser:"
'
'LaserRxBox
'
Me.LaserRxBox.AcceptsReturn = True
```

```
        Me.LaserRxBox.BackColor = System.Drawing.SystemColors.Window
        Me.LaserRxBox.Font = New System.Drawing.Font("Lucida Console", 9.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,
Byte))
        Me.LaserRxBox.Location = New System.Drawing.Point(6, 166)
        Me.LaserRxBox.MaximumSize = New System.Drawing.Size(284, 121)
        Me.LaserRxBox.MaxLength = 8192
        Me.LaserRxBox.MinimumSize = New System.Drawing.Size(284, 121)
        Me.LaserRxBox.Multiline = True
        Me.LaserRxBox.Name = "LaserRxBox"
        Me.LaserRxBox.ReadOnly = True
        Me.LaserRxBox.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.LaserRxBox.Size = New System.Drawing.Size(284, 121)
        Me.LaserRxBox.TabIndex = 16
        Me.LaserRxBox.Text = "Text Received from the Laser:  " &
Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10) &
Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10)
        '
        'AutomaticPulsingTrue
        '
        Me.AutomaticPulsingTrue.AutoSize = True
        Me.AutomaticPulsingTrue.Location = New System.Drawing.Point(140, 17)
        Me.AutomaticPulsingTrue.MaximumSize = New System.Drawing.Size(43, 17)
        Me.AutomaticPulsingTrue.MinimumSize = New System.Drawing.Size(43, 17)
        Me.AutomaticPulsingTrue.Name = "AutomaticPulsingTrue"
        Me.AutomaticPulsingTrue.Size = New System.Drawing.Size(43, 17)
        Me.AutomaticPulsingTrue.TabIndex = 1
        Me.AutomaticPulsingTrue.Text = "Yes"
        Me.AutomaticPulsingTrue.UseVisualStyleBackColor = True
        '
        'ButtonInsertComment
        '
        Me.ButtonInsertComment.Location = New System.Drawing.Point(314, 371)
        Me.ButtonInsertComment.MaximumSize = New System.Drawing.Size(134, 23)
        Me.ButtonInsertComment.MinimumSize = New System.Drawing.Size(134, 23)
        Me.ButtonInsertComment.Name = "ButtonInsertComment"
        Me.ButtonInsertComment.Size = New System.Drawing.Size(134, 23)
        Me.ButtonInsertComment.TabIndex = 18
        Me.ButtonInsertComment.Text = "Insert Comment"
        Me.ButtonInsertComment.UseVisualStyleBackColor = True
        '
        'ButtonEnd
        '
        Me.ButtonEnd.Location = New System.Drawing.Point(535, 371)
        Me.ButtonEnd.MaximumSize = New System.Drawing.Size(75, 23)
        Me.ButtonEnd.MinimumSize = New System.Drawing.Size(75, 23)
        Me.ButtonEnd.Name = "ButtonEnd"
        Me.ButtonEnd.Size = New System.Drawing.Size(75, 23)
        Me.ButtonEnd.TabIndex = 20
        Me.ButtonEnd.Text = "End"
        Me.ButtonEnd.UseVisualStyleBackColor = True
        '
        'ButtonHelp
        '
        Me.ButtonHelp.Location = New System.Drawing.Point(454, 371)
        Me.ButtonHelp.MaximumSize = New System.Drawing.Size(75, 23)
        Me.ButtonHelp.MinimumSize = New System.Drawing.Size(75, 23)
        Me.ButtonHelp.Name = "ButtonHelp"
        Me.ButtonHelp.Size = New System.Drawing.Size(75, 23)
```

```
        Me.ButtonHelp.TabIndex = 19
        Me.ButtonHelp.Text = "Help"
        Me.ButtonHelp.UseVisualStyleBackColor = True
        '
        'LogTrace
        '
        Me.LogTrace.AcceptsReturn = True
        Me.LogTrace.BackColor = System.Drawing.SystemColors.Window
        Me.LogTrace.Font = New System.Drawing.Font("Lucida Console", 9.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,
Byte))
        Me.LogTrace.Location = New System.Drawing.Point(12, 238)
        Me.LogTrace.MaximumSize = New System.Drawing.Size(296, 156)
        Me.LogTrace.MaxLength = 8192
        Me.LogTrace.MinimumSize = New System.Drawing.Size(296, 156)
        Me.LogTrace.Multiline = True
        Me.LogTrace.Name = "LogTrace"
        Me.LogTrace.ReadOnly = True
        Me.LogTrace.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.LogTrace.Size = New System.Drawing.Size(296, 156)
        Me.LogTrace.TabIndex = 17
        Me.LogTrace.Text = "Output Log Follows:  " &
Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10) &
Global.Microsoft.VisualBasic.ChrW(13) & Global.Microsoft.VisualBasic.ChrW(10)
        '
        'Label20
        '
        Me.Label20.AutoSize = True
        Me.Label20.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.Label20.Location = New System.Drawing.Point(12, 222)
        Me.Label20.MaximumSize = New System.Drawing.Size(60, 13)
        Me.Label20.MinimumSize = New System.Drawing.Size(60, 13)
        Me.Label20.Name = "Label20"
        Me.Label20.Size = New System.Drawing.Size(60, 13)
        Me.Label20.TabIndex = 16
        Me.Label20.Text = "Output:"
        '
        'RunBox
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(622, 410)
        Me.Controls.Add(Me.Label20)
        Me.Controls.Add(Me.LogTrace)
        Me.Controls.Add(Me.ButtonHelp)
        Me.Controls.Add(Me.ButtonInsertComment)
        Me.Controls.Add(Me.ButtonEnd)
        Me.Controls.Add(Me.GroupBox2)
        Me.Controls.Add(Me.GroupBox1)
        Me.Controls.Add(Me.Label13)
        Me.Controls.Add(Me.CurrentStatusBox)
        Me.Controls.Add(Me.Label11)
        Me.Controls.Add(Me.SpeedInRangeBox)
        Me.Controls.Add(Me.Label9)
        Me.Controls.Add(Me.CavitationBox)
        Me.Controls.Add(Me.Label7)
        Me.Controls.Add(Me.OutputFileBox)
        Me.Controls.Add(Me.Label5)
        Me.Controls.Add(Me.CurrentSpeedBox)
```

```
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.DateBox)
        Me.Controls.Add(Me.TimeBox)
        Me.ForeColor = System.Drawing.SystemColors.WindowText
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.MaximumSize = New System.Drawing.Size(632, 440)
        Me.MinimizeBox = False
        Me.MinimumSize = New System.Drawing.Size(632, 440)
        Me.Name = "RunBox"
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "RUN RUN RUN!!!"
        Me.GroupBox1.ResumeLayout(False)
        Me.GroupBox1.PerformLayout()
        Me.GroupBox2.ResumeLayout(False)
        Me.GroupBox2.PerformLayout()
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents TimeBox As System.Windows.Forms.Label
    Friend WithEvents DateBox As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents Label5 As System.Windows.Forms.Label
    Friend WithEvents CurrentSpeedBox As System.Windows.Forms.Label
    Friend WithEvents Label7 As System.Windows.Forms.Label
    Friend WithEvents OutputFileBox As System.Windows.Forms.Label
    Friend WithEvents Label9 As System.Windows.Forms.Label
    Friend WithEvents CavitationBox As System.Windows.Forms.Label
    Friend WithEvents Label11 As System.Windows.Forms.Label
    Friend WithEvents SpeedInRangeBox As System.Windows.Forms.Label
    Friend WithEvents Label13 As System.Windows.Forms.Label
    Friend WithEvents CurrentStatusBox As System.Windows.Forms.Label
    Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
    Friend WithEvents GroupBox2 As System.Windows.Forms.GroupBox
    Friend WithEvents OnCavitationContinue As System.Windows.Forms.RadioButton
    Friend WithEvents OnCavitationStop As System.Windows.Forms.RadioButton
    Friend WithEvents ButtonInsertComment As System.Windows.Forms.Button
    Friend WithEvents ButtonEnd As System.Windows.Forms.Button
    Friend WithEvents ButtonHelp As System.Windows.Forms.Button
    Friend WithEvents LogTrace As System.Windows.Forms.TextBox
    Friend WithEvents ButtonStopSpeed As System.Windows.Forms.Button
    Friend WithEvents ButtonStartSpeed As System.Windows.Forms.Button
    Friend WithEvents ButtonSetSpeed As System.Windows.Forms.Button
    Friend WithEvents OnCavitationStopAndRestart As
System.Windows.Forms.RadioButton
    Friend WithEvents AutomaticPulsingTrue As System.Windows.Forms.RadioButton
    Friend WithEvents SetSpeedInputBox As System.Windows.Forms.TextBox
    Friend WithEvents Label15 As System.Windows.Forms.Label
    Friend WithEvents Label17 As System.Windows.Forms.Label
    Friend WithEvents SetSpeedShowBox As System.Windows.Forms.Label
    Friend WithEvents Label18 As System.Windows.Forms.Label
    Friend WithEvents RestartDelayShowBox As System.Windows.Forms.Label
    Friend WithEvents RestartDelayInputBox As System.Windows.Forms.TextBox
    Friend WithEvents Label20 As System.Windows.Forms.Label
    Friend WithEvents Label22 As System.Windows.Forms.Label
```

```
    Friend WithEvents AutomaticPulsingFalse As System.Windows.Forms.RadioButton
    Friend WithEvents Label21 As System.Windows.Forms.Label
    Friend WithEvents LaserRxBox As System.Windows.Forms.TextBox
    Friend WithEvents ButtonFireLaser As System.Windows.Forms.Button
    Friend WithEvents ButtonSendLaserCommand As System.Windows.Forms.Button
    Friend WithEvents FireLaserBox As System.Windows.Forms.Label
    Friend WithEvents ButtonSetLaser As System.Windows.Forms.Button
    Friend WithEvents Label26 As System.Windows.Forms.Label
    Friend WithEvents EstimatedDurationShowBox As System.Windows.Forms.Label
    Friend WithEvents EstimatedDurationInputBox As System.Windows.Forms.TextBox
    Friend WithEvents Label23 As System.Windows.Forms.Label
    Friend WithEvents LaserDelayShowBox As System.Windows.Forms.Label
    Friend WithEvents LaserDelayInputBox As System.Windows.Forms.TextBox
    Friend WithEvents Label28 As System.Windows.Forms.Label
    Friend WithEvents NextPulsePowerShowBox As System.Windows.Forms.Label
End Class
```

B.12 RUNBOX.VB

```
Public Class RunBox
    'Code for the RunBox form -- The primary interface between the Code and the
User that is shown during normal operation



    Private Sub RunBox_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        'Stuff to do when the form is closing

        If e.CloseReason <> CloseReason.None Then
            e.Cancel = True
        End If

    End Sub



    Private Sub RunBox_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Stuff to do when the form is loaded
        'set the defaults

        'set the title
        Me.Text = My.Application.Info.ProductName & " v. " &
My.Application.Info.Version.Major & "." & My.Application.Info.Version.Minor &
"." & My.Application.Info.Version.Build & "." &
My.Application.Info.Version.Revision

        'set default values for the speed control side
        OnCavitationStop.Checked = True
        RestartDelayInputBox.Text = "5"
        RestartDelayShowBox.Text = "5"
        SetSpeedInputBox.Text = "0"
        SetSpeedShowBox.Text = "0"

        'set default values for the laser control side
        NextPulsePowerShowBox.Text = "1"
```

```
        EstimatedDurationInputBox.Text = "0.00001"
        EstimatedDurationShowBox.Text = "0.00001"
        LaserDelayInputBox.Text = "2.5"
        LaserDelayShowBox.Text = "2.5"
        AutomaticPulsingFalse.Checked = True

        'send the values to the shared variables
        SyncLock Windowsill.RunBoxLock
            'load laser control values
            Windowsill.NextLaserPulsePower = Val(NextPulsePowerShowBox.Text)
            Windowsill.EstimatedLaserPulseDuration =
Val(EstimatedDurationShowBox.Text)
            Windowsill.DelayBetweenPulses = Val(LaserDelayShowBox.Text)
            Windowsill.AutomaticPulsing = AutomaticPulsingTrue.Checked
            'load speed control values
            Windowsill.RunButtonClicked = False
            Windowsill.DesiredSpeed = Val(SetSpeedShowBox.Text)
            Windowsill.RestartDelayTime = Val(RestartDelayShowBox.Text)
            If OnCavitationContinue.Checked = True Then
                'continue on cavitation
                Windowsill.CavitationAction = 0
            ElseIf OnCavitationStop.Checked = True Then
                'stop
                Windowsill.CavitationAction = 1
            ElseIf OnCavitationStopAndRestart.Checked = True Then
                'stop and restart
                Windowsill.CavitationAction = 2
            Else
                '?
                Windowsill.CavitationAction = 3
            End If
        End SyncLock

    End Sub



    Private Sub ButtonInsertComment_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles ButtonInsertComment.Click
        'User wants to Insert a Comment into the Logfile

        InsertComment.Show()
        InsertComment.BringToFront()

    End Sub



    Private Sub ButtonHelp_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonHelp.Click
        'User thinks help may be available
        'HAHAHAHAHAHA

        HelpBox.Show()
        HelpBox.BringToFront()

    End Sub
```

```
    Private Sub ButtonEnd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonEnd.Click
        'User click on the END button

        Dim LocalExit As Boolean

        'make sure a shutdown is not already in progress
        SyncLock CentralClass.ProgramExitLock
            LocalExit = CentralClass.ExitProgram
        End SyncLock
        If LocalExit = False Then
            'make sure User wants to Exit
            If MsgBox("Do you really want to End the Program?",
MsgBoxStyle.YesNo, "End it all?") = MsgBoxResult.Yes Then
                SyncLock CentralClass.ProgramExitLock
                    CentralClass.ExitProgram = True
                End SyncLock
                Lumberjack.SendToLog("Program Shutdown Initiated by User")
            End If
        End If
        Application.DoEvents()

    End Sub



    Private Sub ButtonFireLaser_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonFireLaser.Click
        'fire a user-defined pulse from the laser
        Dim FormerState As Boolean

        'first disable automatic pulsing
        SyncLock Windowsill.RunBoxLock
            FormerState = Windowsill.AutomaticPulsing
            Windowsill.AutomaticPulsing = False
            Me.AutomaticPulsingFalse.Checked = True
        End SyncLock
        'log the event
        If FormerState = True Then
            Lumberjack.SendToLog("Disable Automatic Laser Pulsing")
        End If

        'then show the pulse box
        FirePulse.Show()
        FirePulse.BringToFront()
        Application.DoEvents()

    End Sub



    Private Sub ButtonSendLaserCommand_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ButtonSendLaserCommand.Click
        'Send a text command to the laser
        Dim FormerState As Boolean

        'first disable automatic pulsing
        SyncLock Windowsill.RunBoxLock
```

```
            FormerState = Windowsill.AutomaticPulsing
            Windowsill.AutomaticPulsing = False
            Me.AutomaticPulsingFalse.Checked = True
        End SyncLock
        'log the event
        If FormerState = True Then
            Lumberjack.SendToLog("Disable Automatic Laser Pulsing")
        End If

        'then show the command box
        SendLaserCommand.Show()
        SendLaserCommand.BringToFront()
        Application.DoEvents()

    End Sub



    Private Sub AutomaticPulsingFalse_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AutomaticPulsingFalse.Click
        'immediately disabele automatic pulsing
        Dim FormerState As Boolean

        SyncLock Windowsill.RunBoxLock
            FormerState = Windowsill.AutomaticPulsing
            Windowsill.AutomaticPulsing = False
            Me.AutomaticPulsingFalse.Checked = True
        End SyncLock
        'log the event
        If FormerState = True Then
            Lumberjack.SendToLog("Disable Automatic Laser Pulsing")
        End If
        Application.DoEvents()

    End Sub



    Private Sub ButtonStartSpeed_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ButtonStartSpeed.Click
        'user pushed start

        'alert the program
        SyncLock Windowsill.RunBoxLock
            Windowsill.RunButtonClicked = True
        End SyncLock

        'disable the start button
        Me.ButtonStartSpeed.Enabled = False
        'enable the stop button
        Me.ButtonStopSpeed.Enabled = True
        'log the event
        Lumberjack.SendToLog("Start Automatically Controlling the Speed")
        Application.DoEvents()

    End Sub
```

```
    Private Sub ButtonStopSpeed_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonStopSpeed.Click
        'something pushed stop

        Dim OldStatus As Boolean
        Dim ShuttingDown As Boolean

        'alert the program
        SyncLock Windowsill.RunBoxLock
            OldStatus = Windowsill.RunButtonClicked
            Windowsill.RunButtonClicked = False
        End SyncLock

        'disable the stop button
        Me.ButtonStopSpeed.Enabled = False
        'enable the start button, unless shutting down
        Me.ButtonStartSpeed.Enabled = True
        'get shutdown status
        SyncLock CentralClass.ProgramExitLock
            ShuttingDown = CentralClass.ExitProgram
        End SyncLock
        If ShuttingDown = True Then
            'disable button if shutting down
            Me.ButtonStartSpeed.Enabled = False
        End If
        Application.DoEvents()
        'log the event, unless nothing changed
        If OldStatus = True Then
            Lumberjack.SendToLog("Stop Automatically Controlling the Speed")
        End If
        'stop autopulsing
        Call StopAutomaticPulsing()

    End Sub



    Friend Sub StopAutomaticPulsing()
        'call from main thread

        Call AutomaticPulsingFalse_Click(Me, EventArgs.Empty)

    End Sub



    Friend Sub PressStopButton()
        'call from main thread

        Call ButtonStopSpeed_Click(Me, EventArgs.Empty)

    End Sub



    Private Sub ButtonSetSpeed_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonSetSpeed.Click
        'sets the speed control values
```

```
        Try

            Dim DesiredSpeedChanged As Boolean = False
            Dim NewDesiredSpeed As Double = 0
            Dim RestartDelayTimeChanged As Boolean = False
            Dim NewDelayTime As Double = 0
            Dim CavitationActionChanged As Boolean = False
            Dim NewCavitationAction As Long = 0
            Dim UpdateSpeed As Boolean = False
            Dim UpdateDelay As Boolean = False
            Dim DelayHolder As Double = -1


            'load delay time value
            Try
                DelayHolder = Val(RestartDelayInputBox.Text)
                UpdateDelay = True
            Catch BadDelayValue As OverflowException
                'bad value for new delay time
                UpdateDelay = False
                DelayHolder = -1
                MsgBox("Illegal Value for the new Delay Time" & vbNewLine &
RestartDelayInputBox.Text, , "Overflow")
            End Try

            'send the values to the shared variables
            If DelayHolder >= 0 Then
                NewDelayTime = DelayHolder
                'load speed control values
                Try
                    NewDesiredSpeed = Val(SetSpeedInputBox.Text)
                    UpdateSpeed = True
                Catch TooMuchSpeed As OverflowException
                    'bad value for speed
                    UpdateSpeed = False
                    MsgBox("Illegal Value for the new Speed" & vbNewLine &
SetSpeedInputBox.Text, , "Overflow")
                End Try

                'cavitation action
                If OnCavitationContinue.Checked = True Then
                    'continue on cavitation
                    NewCavitationAction = 0
                ElseIf OnCavitationStop.Checked = True Then
                    'stop
                    NewCavitationAction = 1
                ElseIf OnCavitationStopAndRestart.Checked = True Then
                    'stop and restart
                    NewCavitationAction = 2
                Else
                    '?
                    NewCavitationAction = 3
                End If
                SyncLock Windowsill.RunBoxLock
                    'commit changes
                    If NewCavitationAction <> Windowsill.CavitationAction Then
                        'action on cavitation changed
                        Windowsill.CavitationAction = NewCavitationAction
                        CavitationActionChanged = True
```

```
                    End If
                    If UpdateSpeed = True Then
                        If NewDesiredSpeed <> Windowsill.DesiredSpeed Then
                            'desired speed changed
                            Windowsill.DesiredSpeed = NewDesiredSpeed
                            DesiredSpeedChanged = True
                        End If
                    End If
                    If UpdateDelay = True Then
                        If NewDelayTime <> Windowsill.RestartDelayTime Then
                            'restart delay changed
                            Windowsill.RestartDelayTime = NewDelayTime
                            RestartDelayTimeChanged = True
                        End If
                    End If
                End SyncLock
            Else
                Beep()
                MsgBox("Restart Delays must be >=0, no values changed")
            End If

            'now log any changes
            If DesiredSpeedChanged = True Then
                Lumberjack.SendToLog("Desired Control Speed changed to " &
Str(NewDesiredSpeed))
            End If
            If RestartDelayTimeChanged = True Then
                Lumberjack.SendToLog("Restart Delay Time changed to " &
Str(NewDelayTime))
            End If
            If CavitationActionChanged = True Then
                If NewCavitationAction = 0 Then
                    Lumberjack.SendToLog("Action on cavitation changed to
Continue")
                ElseIf NewCavitationAction = 1 Then
                    Lumberjack.SendToLog("Action on cavitation changed to
Stop")
                ElseIf NewCavitationAction = 2 Then
                    Lumberjack.SendToLog("Action on cavitation changed to Stop
+ Restart")
                Else
                    Lumberjack.SendToLog("Action on cavitation changed to
?????")
                End If
            End If

            Application.DoEvents()

        Catch TooBig As OverflowException
            'something overflowed
            Lumberjack.SendToLog("Recoverable Exception in " &
Thread.CurrentThread.Name & ":  " & TooBig.Message & vbNewLine & "Details:  " &
vbNewLine & TooBig.ToString)
            MsgBox("An Overflow Exception has occurred in " &
Thread.CurrentThread.Name & ", but execution will continue.  Details:  " &
vbNewLine & TooBig.ToString, , "OUCH!")
        Catch BigException As Exception
            'something got really screwed up
```

```
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End Try

    End Sub



    Private Sub ButtonSetLaser_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ButtonSetLaser.Click
        'sets the laser control values

        Try

            Dim ChangedPulseDuration As Boolean = False
            Dim NewPulseDuration As Double = 0
            Dim ChangedPulseDelay As Boolean = False
            Dim NewPulseDelay As Double = 0
            Dim ChangedAutoPulsing As Boolean = False
            Dim NewAutoPulsing As Boolean = False
            Dim DurationHolder As Double = -1
            Dim DelayHolder As Double = -1

            'test the values
            Try
                DurationHolder = Val(EstimatedDurationInputBox.Text)
            Catch TooMuchSpeed As OverflowException
                'bad value for duration
                MsgBox("Illegal Value for the new Estimated Duration:" &
vbNewLine & EstimatedDurationInputBox.Text, , "Overflow")
                DurationHolder = -1
            End Try

            Try
                DelayHolder = Val(LaserDelayInputBox.Text)
            Catch TooMuchSpeed As OverflowException
                'bad value for duration
                MsgBox("Illegal Value for the new Delay Between Pulses:" &
vbNewLine & LaserDelayInputBox.Text, , "Overflow")
                DelayHolder = -1
            End Try

            'sanity check
            If DurationHolder >= 0 Then
                'it works so far
                NewPulseDuration = DurationHolder
                If DelayHolder > 0 Then
                    'both work
                    NewPulseDelay = DelayHolder
                    NewAutoPulsing = AutomaticPulsingTrue.Checked
                    SyncLock Windowsill.RunBoxLock
                        'load laser control values
```

```
                        If Windowsill.EstimatedLaserPulseDuration <>
NewPulseDuration Then
                            ChangedPulseDuration = True
                            Windowsill.EstimatedLaserPulseDuration =
NewPulseDuration
                        End If
                        If Windowsill.DelayBetweenPulses <> NewPulseDelay Then
                            ChangedPulseDelay = True
                            Windowsill.DelayBetweenPulses = NewPulseDelay
                        End If
                        If Windowsill.AutomaticPulsing <> NewAutoPulsing Then
                            ChangedAutoPulsing = True
                            Windowsill.AutomaticPulsing = NewAutoPulsing
                        End If
                    End SyncLock
                Else
                    MsgBox("Delay between laser pulses must be >0")
                End If
            Else
                MsgBox("Estimated Pulse Duration values must be >=0")
            End If

            'log the changes
            If ChangedPulseDuration = True Then
                Lumberjack.SendToLog("Estimated Laser Pulse Duration changed to
" & Str(NewPulseDuration))
            End If
            If ChangedPulseDelay = True Then
                Lumberjack.SendToLog("Changed Delay between Laser Pulses to " &
Str(NewPulseDelay))
            End If
            If ChangedAutoPulsing = True Then
                If NewAutoPulsing = True Then
                    Lumberjack.SendToLog("Changed to Automatic Laser Pulsing")
                Else
                    Lumberjack.SendToLog("Changed to Manual Laser Pulsing")
                End If
            End If
            Application.DoEvents()

        Catch TooBig As OverflowException
            'something overflowed
            Lumberjack.SendToLog("Recoverable Exception in " &
Thread.CurrentThread.Name & ":  " & TooBig.Message & vbNewLine & "Details:  " &
vbNewLine & TooBig.ToString)
            MsgBox("An Overflow Exception has occurred in " &
Thread.CurrentThread.Name & ", but execution will continue.  Details:  " &
vbNewLine & TooBig.ToString, , "OUCH!")
        Catch BigException As Exception
            'something got really screwed up
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
```

```
        End Try

    End Sub


End Class
```

B.13 SENDLASERCOMMAND.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class SendLaserCommand
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(SendLaserCommand))
        Me.CommandText = New System.Windows.Forms.TextBox
        Me.CancelCommandButton = New System.Windows.Forms.Button
        Me.SendCommandButton = New System.Windows.Forms.Button
        Me.CommandNoteBox = New System.Windows.Forms.Label
        Me.NewlineButton = New System.Windows.Forms.Button
        Me.SuspendLayout()
        '
        'CommandText
        '
        Me.CommandText.AcceptsReturn = True
        Me.CommandText.Location = New System.Drawing.Point(12, 12)
        Me.CommandText.MaximumSize = New System.Drawing.Size(270, 206)
        Me.CommandText.MinimumSize = New System.Drawing.Size(270, 206)
        Me.CommandText.Multiline = True
        Me.CommandText.Name = "CommandText"
        Me.CommandText.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.CommandText.Size = New System.Drawing.Size(270, 206)
        Me.CommandText.TabIndex = 0
        '
        'CancelCommandButton
        '
```

```
        Me.CancelCommandButton.Location = New System.Drawing.Point(207, 224)
        Me.CancelCommandButton.MaximumSize = New System.Drawing.Size(75, 23)
        Me.CancelCommandButton.MinimumSize = New System.Drawing.Size(75, 23)
        Me.CancelCommandButton.Name = "CancelCommandButton"
        Me.CancelCommandButton.Size = New System.Drawing.Size(75, 23)
        Me.CancelCommandButton.TabIndex = 2
        Me.CancelCommandButton.Text = "Cancel"
        Me.CancelCommandButton.UseVisualStyleBackColor = True
        '
        'SendCommandButton
        '
        Me.SendCommandButton.Location = New System.Drawing.Point(12, 224)
        Me.SendCommandButton.MaximumSize = New System.Drawing.Size(75, 23)
        Me.SendCommandButton.MinimumSize = New System.Drawing.Size(75, 23)
        Me.SendCommandButton.Name = "SendCommandButton"
        Me.SendCommandButton.Size = New System.Drawing.Size(75, 23)
        Me.SendCommandButton.TabIndex = 1
        Me.SendCommandButton.Text = "OK"
        Me.SendCommandButton.UseVisualStyleBackColor = True
        '
        'CommandNoteBox
        '
        Me.CommandNoteBox.AutoSize = True
        Me.CommandNoteBox.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.CommandNoteBox.FlatStyle = System.Windows.Forms.FlatStyle.Flat
        Me.CommandNoteBox.Location = New System.Drawing.Point(12, 250)
        Me.CommandNoteBox.MaximumSize = New System.Drawing.Size(270, 15)
        Me.CommandNoteBox.MinimumSize = New System.Drawing.Size(270, 15)
        Me.CommandNoteBox.Name = "CommandNoteBox"
        Me.CommandNoteBox.Size = New System.Drawing.Size(270, 15)
        Me.CommandNoteBox.TabIndex = 3
        '
        'NewlineButton
        '
        Me.NewlineButton.Location = New System.Drawing.Point(99, 224)
        Me.NewlineButton.MaximumSize = New System.Drawing.Size(96, 23)
        Me.NewlineButton.MinimumSize = New System.Drawing.Size(96, 23)
        Me.NewlineButton.Name = "NewlineButton"
        Me.NewlineButton.Size = New System.Drawing.Size(96, 23)
        Me.NewlineButton.TabIndex = 4
        Me.NewlineButton.Text = "Insert EOL"
        Me.NewlineButton.UseVisualStyleBackColor = True
        '
        'SendLaserCommand
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(294, 268)
        Me.Controls.Add(Me.NewlineButton)
        Me.Controls.Add(Me.CommandNoteBox)
        Me.Controls.Add(Me.CommandText)
        Me.Controls.Add(Me.CancelCommandButton)
        Me.Controls.Add(Me.SendCommandButton)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.MaximumSize = New System.Drawing.Size(300, 300)
        Me.MinimizeBox = False
```

```
        Me.MinimumSize = New System.Drawing.Size(300, 300)
        Me.Name = "SendLaserCommand"
        Me.ShowIcon = False
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "Send Command to Laser"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents CommandText As System.Windows.Forms.TextBox
    Friend WithEvents CancelCommandButton As System.Windows.Forms.Button
    Friend WithEvents SendCommandButton As System.Windows.Forms.Button
    Friend WithEvents CommandNoteBox As System.Windows.Forms.Label
    Friend WithEvents NewlineButton As System.Windows.Forms.Button
End Class
```

B.14 SENDLASERCOMMAND.VB

```
Public Class SendLaserCommand
    'Code for the SendLaserCommand form -- the form that shows when the User
wishes to manually send a string of text to the laser



    Private Shared SendingCommand As Boolean = False
    Private Shared SendLockObj As New Object



    Private Sub SendLaserCommand_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        'Stuff to do when the form is closing

        If e.CloseReason <> CloseReason.None Then
            e.Cancel = True
        End If

    End Sub



    Private Sub CancelCommandButton_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CancelCommandButton.Click
        'cancel any action and close the box

        CommandText.Text = ""
        CommandNoteBox.Text = ""
        Me.Hide()

    End Sub



    Private Sub SendCommandButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles SendCommandButton.Click
        'send any text to the laser
```

```
        Dim TextToSend As String = ""
        Dim DatalineLockdown As Boolean = False
        Dim InUse As Boolean = False

        'only one send at a time
        SyncLock SendLockObj
            InUse = SendingCommand
            If SendingCommand = False Then
                'indicate that the thing is in use
                SendingCommand = True
            End If
        End SyncLock

        If InUse = False Then
            'we can proceed; there is not an overlapping attempt
            'get the command and clear the box
            TextToSend = CommandText.Text
            CommandText.Text = ""
            CommandNoteBox.Text = ""
            Me.Hide()

            SyncLock Windowsill.RunBoxLock
                'only xmit if the line is clear
                DatalineLockdown = Windowsill.LaserTXDataCritical
                If DatalineLockdown = False Then
                    'copy the text
                    Windowsill.CommandToLaser = Windowsill.CommandToLaser &
TextToSend
                End If
            End SyncLock

            'If the line was not clear, restore the box and show the note
            If DatalineLockdown = True Then
                'restore the box
                Me.Show()
                'show an explanatory note
                CommandNoteBox.Text = "Could Not Proceed, Try Again"
                'restore the command text
                CommandText.Text = TextToSend
                CommandText.Select(CommandText.TextLength, 0)
                CommandText.ScrollToCaret()
                Beep()
            Else
                'the line was open, log the transmission
                Lumberjack.SendToLog("The Following User-Issued Command Will Be
Sent To The Laser:  " & TextToSend)
            End If

            'unblock the line
            SyncLock SendLockObj
                SendingCommand = False
            End SyncLock

        End If

    End Sub
```

```
    Private Sub NewlineButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles NewlineButton.Click
        'inserts a newline character specific to the laser at the caret
location

        Dim EOLString As String
        Dim EditString As String
        Dim CursorLocation As Integer
        Dim SelectedLength As Integer


        'get the newline sequence
        SyncLock CentralClass.FileOpsLock
            EOLString = CentralClass.LaserNewlineString
        End SyncLock

        SyncLock SendLockObj
            'only proceed if a send is not currently being issued
            'lock down the text box
            CommandText.ReadOnly = True
            'get the text
            EditString = CommandText.Text
            'get the caret or selection begin location and selection length
            CursorLocation = CommandText.SelectionStart
            SelectedLength = CommandText.SelectionLength
            'insert the eol sequence
            CommandText.Text = EditString.Insert(CursorLocation, EOLString)
            'shift the selection
            CommandText.SelectionStart = CursorLocation + 1
            CommandText.SelectionLength = SelectedLength
            'unlock the box
            CommandText.ReadOnly = False
        End SyncLock

    End Sub



End Class
```

B.15 STARTUPFORM.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class StartupForm
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub
```

```
    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(StartupForm))
        Me.OKButton = New System.Windows.Forms.Button
        Me.ProgramHelpButton = New System.Windows.Forms.Button
        Me.ExitButton = New System.Windows.Forms.Button
        Me.FilenameBox = New System.Windows.Forms.TextBox
        Me.RadioButtonAppend = New System.Windows.Forms.RadioButton
        Me.RadioButtonReplace = New System.Windows.Forms.RadioButton
        Me.BrowseButton = New System.Windows.Forms.Button
        Me.Panel1 = New System.Windows.Forms.Panel
        Me.Label1 = New System.Windows.Forms.Label
        Me.Label2 = New System.Windows.Forms.Label
        Me.Label3 = New System.Windows.Forms.Label
        Me.ControllerPortComboBox = New System.Windows.Forms.ComboBox
        Me.LaserPortComboBox = New System.Windows.Forms.ComboBox
        Me.PortSpeedComboBox = New System.Windows.Forms.ComboBox
        Me.Label4 = New System.Windows.Forms.Label
        Me.TimingCheckBox = New System.Windows.Forms.CheckBox
        Me.Panel1.SuspendLayout()
        Me.SuspendLayout()
        '
        'OKButton
        '
        Me.OKButton.Location = New System.Drawing.Point(46, 256)
        Me.OKButton.MaximumSize = New System.Drawing.Size(128, 40)
        Me.OKButton.MinimumSize = New System.Drawing.Size(128, 40)
        Me.OKButton.Name = "OKButton"
        Me.OKButton.Size = New System.Drawing.Size(128, 40)
        Me.OKButton.TabIndex = 7
        Me.OKButton.Text = "OK"
        Me.OKButton.UseVisualStyleBackColor = True
        '
        'ProgramHelpButton
        '
        Me.ProgramHelpButton.Location = New System.Drawing.Point(211, 256)
        Me.ProgramHelpButton.MaximumSize = New System.Drawing.Size(128, 40)
        Me.ProgramHelpButton.MinimumSize = New System.Drawing.Size(128, 40)
        Me.ProgramHelpButton.Name = "ProgramHelpButton"
        Me.ProgramHelpButton.Size = New System.Drawing.Size(128, 40)
        Me.ProgramHelpButton.TabIndex = 8
        Me.ProgramHelpButton.Text = "Help"
        Me.ProgramHelpButton.UseVisualStyleBackColor = True
        '
        'ExitButton
        '
        Me.ExitButton.Location = New System.Drawing.Point(368, 256)
        Me.ExitButton.MaximumSize = New System.Drawing.Size(128, 40)
        Me.ExitButton.MinimumSize = New System.Drawing.Size(128, 40)
        Me.ExitButton.Name = "ExitButton"
        Me.ExitButton.Size = New System.Drawing.Size(128, 40)
```

```
Me.ExitButton.TabIndex = 9
Me.ExitButton.Text = "Exit"
Me.ExitButton.UseVisualStyleBackColor = True
'
'FilenameBox
'
Me.FilenameBox.Location = New System.Drawing.Point(22, 95)
Me.FilenameBox.MaximumSize = New System.Drawing.Size(406, 20)
Me.FilenameBox.MinimumSize = New System.Drawing.Size(406, 20)
Me.FilenameBox.Name = "FilenameBox"
Me.FilenameBox.Size = New System.Drawing.Size(406, 20)
Me.FilenameBox.TabIndex = 4
Me.FilenameBox.Text = "output.txt"
'
'RadioButtonAppend
'
Me.RadioButtonAppend.AutoSize = True
Me.RadioButtonAppend.Checked = True
Me.RadioButtonAppend.Location = New System.Drawing.Point(194, 51)
Me.RadioButtonAppend.MaximumSize = New System.Drawing.Size(62, 17)
Me.RadioButtonAppend.MinimumSize = New System.Drawing.Size(62, 17)
Me.RadioButtonAppend.Name = "RadioButtonAppend"
Me.RadioButtonAppend.Size = New System.Drawing.Size(62, 17)
Me.RadioButtonAppend.TabIndex = 2
Me.RadioButtonAppend.TabStop = True
Me.RadioButtonAppend.Text = "Append"
Me.RadioButtonAppend.UseVisualStyleBackColor = True
'
'RadioButtonReplace
'
Me.RadioButtonReplace.AutoSize = True
Me.RadioButtonReplace.Location = New System.Drawing.Point(363, 51)
Me.RadioButtonReplace.MaximumSize = New System.Drawing.Size(65, 17)
Me.RadioButtonReplace.MinimumSize = New System.Drawing.Size(65, 17)
Me.RadioButtonReplace.Name = "RadioButtonReplace"
Me.RadioButtonReplace.Size = New System.Drawing.Size(65, 17)
Me.RadioButtonReplace.TabIndex = 3
Me.RadioButtonReplace.Text = "Replace"
Me.RadioButtonReplace.UseVisualStyleBackColor = True
'
'BrowseButton
'
Me.BrowseButton.Location = New System.Drawing.Point(22, 46)
Me.BrowseButton.MaximumSize = New System.Drawing.Size(86, 26)
Me.BrowseButton.MinimumSize = New System.Drawing.Size(86, 26)
Me.BrowseButton.Name = "BrowseButton"
Me.BrowseButton.Size = New System.Drawing.Size(86, 26)
Me.BrowseButton.TabIndex = 1
Me.BrowseButton.Text = "Browse..."
Me.BrowseButton.UseVisualStyleBackColor = True
'
'Panel1
'
Me.Panel1.Controls.Add(Me.Label1)
Me.Panel1.Controls.Add(Me.BrowseButton)
Me.Panel1.Controls.Add(Me.RadioButtonReplace)
Me.Panel1.Controls.Add(Me.RadioButtonAppend)
Me.Panel1.Controls.Add(Me.FilenameBox)
Me.Panel1.Location = New System.Drawing.Point(46, 97)
```

```
        Me.Panel1.MaximumSize = New System.Drawing.Size(450, 128)
        Me.Panel1.MinimumSize = New System.Drawing.Size(450, 128)
        Me.Panel1.Name = "Panel1"
        Me.Panel1.Size = New System.Drawing.Size(450, 128)
        Me.Panel1.TabIndex = 6
        '
        'Label1
        '
        Me.Label1.AutoSize = True
        Me.Label1.Font = New System.Drawing.Font("Microsoft Sans Serif", 16.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,
Byte))
        Me.Label1.Location = New System.Drawing.Point(181, 11)
        Me.Label1.MaximumSize = New System.Drawing.Size(89, 26)
        Me.Label1.MinimumSize = New System.Drawing.Size(89, 26)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(89, 26)
        Me.Label1.TabIndex = 0
        Me.Label1.Text = "Log File"
        '
        'Label2
        '
        Me.Label2.AutoSize = True
        Me.Label2.Font = New System.Drawing.Font("Microsoft Sans Serif", 12.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,
Byte))
        Me.Label2.Location = New System.Drawing.Point(42, 24)
        Me.Label2.MaximumSize = New System.Drawing.Size(128, 20)
        Me.Label2.MinimumSize = New System.Drawing.Size(120, 20)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(120, 20)
        Me.Label2.TabIndex = 0
        Me.Label2.Text = "Controller Port"
        '
        'Label3
        '
        Me.Label3.AutoSize = True
        Me.Label3.Font = New System.Drawing.Font("Microsoft Sans Serif", 12.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,
Byte))
        Me.Label3.Location = New System.Drawing.Point(207, 24)
        Me.Label3.MaximumSize = New System.Drawing.Size(82, 20)
        Me.Label3.MinimumSize = New System.Drawing.Size(82, 20)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(82, 20)
        Me.Label3.TabIndex = 2
        Me.Label3.Text = "Laser Port"
        '
        'ControllerPortComboBox
        '
        Me.ControllerPortComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList
        Me.ControllerPortComboBox.FormattingEnabled = True
        Me.ControllerPortComboBox.Items.AddRange(New Object() {"(none)"})
        Me.ControllerPortComboBox.Location = New System.Drawing.Point(46, 47)
        Me.ControllerPortComboBox.MaximumSize = New System.Drawing.Size(128, 0)
        Me.ControllerPortComboBox.MinimumSize = New System.Drawing.Size(128, 0)
        Me.ControllerPortComboBox.Name = "ControllerPortComboBox"
        Me.ControllerPortComboBox.Size = New System.Drawing.Size(128, 21)
```

```
        Me.ControllerPortComboBox.TabIndex = 1
        '
        'LaserPortComboBox
        '
        Me.LaserPortComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList
        Me.LaserPortComboBox.FormattingEnabled = True
        Me.LaserPortComboBox.Items.AddRange(New Object() {"(none)"})
        Me.LaserPortComboBox.Location = New System.Drawing.Point(211, 47)
        Me.LaserPortComboBox.MaximumSize = New System.Drawing.Size(128, 0)
        Me.LaserPortComboBox.MinimumSize = New System.Drawing.Size(128, 0)
        Me.LaserPortComboBox.Name = "LaserPortComboBox"
        Me.LaserPortComboBox.Size = New System.Drawing.Size(128, 21)
        Me.LaserPortComboBox.TabIndex = 3
        '
        'PortSpeedComboBox
        '
        Me.PortSpeedComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList
        Me.PortSpeedComboBox.FormattingEnabled = True
        Me.PortSpeedComboBox.Items.AddRange(New Object() {"19200"})
        Me.PortSpeedComboBox.Location = New System.Drawing.Point(368, 47)
        Me.PortSpeedComboBox.MaximumSize = New System.Drawing.Size(128, 0)
        Me.PortSpeedComboBox.MinimumSize = New System.Drawing.Size(128, 0)
        Me.PortSpeedComboBox.Name = "PortSpeedComboBox"
        Me.PortSpeedComboBox.Size = New System.Drawing.Size(128, 21)
        Me.PortSpeedComboBox.TabIndex = 5
        '
        'Label4
        '
        Me.Label4.AutoSize = True
        Me.Label4.Font = New System.Drawing.Font("Microsoft Sans Serif", 12.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0,
Byte))
        Me.Label4.Location = New System.Drawing.Point(364, 24)
        Me.Label4.MaximumSize = New System.Drawing.Size(89, 20)
        Me.Label4.MinimumSize = New System.Drawing.Size(89, 20)
        Me.Label4.Name = "Label4"
        Me.Label4.Size = New System.Drawing.Size(89, 20)
        Me.Label4.TabIndex = 4
        Me.Label4.Text = "Port Speed"
        '
        'TimingCheckBox
        '
        Me.TimingCheckBox.AutoSize = True
        Me.TimingCheckBox.Checked = True
        Me.TimingCheckBox.CheckState = System.Windows.Forms.CheckState.Checked
        Me.TimingCheckBox.Location = New System.Drawing.Point(46, 74)
        Me.TimingCheckBox.Name = "TimingCheckBox"
        Me.TimingCheckBox.Size = New System.Drawing.Size(131, 17)
        Me.TimingCheckBox.TabIndex = 10
        Me.TimingCheckBox.Text = "Use Advanced Timing"
        Me.TimingCheckBox.UseVisualStyleBackColor = True
        '
        'StartupForm
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(540, 347)
```

```
        Me.Controls.Add(Me.TimingCheckBox)
        Me.Controls.Add(Me.PortSpeedComboBox)
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.LaserPortComboBox)
        Me.Controls.Add(Me.ControllerPortComboBox)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.Panel1)
        Me.Controls.Add(Me.ExitButton)
        Me.Controls.Add(Me.ProgramHelpButton)
        Me.Controls.Add(Me.OKButton)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.MaximumSize = New System.Drawing.Size(550, 376)
        Me.MinimumSize = New System.Drawing.Size(550, 376)
        Me.Name = "StartupForm"
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "Speed Controller Port and Logfile"
        Me.Panel1.ResumeLayout(False)
        Me.Panel1.PerformLayout()
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents OKButton As System.Windows.Forms.Button
    Friend WithEvents ProgramHelpButton As System.Windows.Forms.Button
    Friend WithEvents ExitButton As System.Windows.Forms.Button
    Friend WithEvents FilenameBox As System.Windows.Forms.TextBox
    Friend WithEvents RadioButtonAppend As System.Windows.Forms.RadioButton
    Friend WithEvents RadioButtonReplace As System.Windows.Forms.RadioButton
    Friend WithEvents BrowseButton As System.Windows.Forms.Button
    Friend WithEvents Panel1 As System.Windows.Forms.Panel
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents ControllerPortComboBox As System.Windows.Forms.ComboBox
    Friend WithEvents LaserPortComboBox As System.Windows.Forms.ComboBox
    Friend WithEvents PortSpeedComboBox As System.Windows.Forms.ComboBox
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents TimingCheckBox As System.Windows.Forms.CheckBox
End Class
```

B.16 STARTUPFORM.VB

```
Public Class StartupForm
    'Code for the StartupForm -- the first form seen by the User on Program
Startup



    Private Sub BrowseButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BrowseButton.Click
        'The User has clicked on the Browse button in the startup form
        'Shows a file dialog to save the output log file

        'Create the SaveFileDialog
```

```
        Dim LogFileDialog1 As New SaveFileDialog
        'Set the windows title
        LogFileDialog1.Title = "Output Log File"
        'Set the File Filters, defaulting to text
        LogFileDialog1.Filter = "Text File (*.txt)|*.txt|All Files (*.*)|*.*"
        LogFileDialog1.FilterIndex = 1
        'Set the initial directory to My Documents
        LogFileDialog1.InitialDirectory =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        'Automatically add an extension matching the filter
        LogFileDialog1.DefaultExt = ""
        LogFileDialog1.AddExtension = True
        LogFileDialog1.CheckFileExists = False
        LogFileDialog1.OverwritePrompt = False
        LogFileDialog1.CheckPathExists = True
        'Show the dialog
        LogFileDialog1.ShowDialog()
        'Copy the filename to the filename textbox
        If LogFileDialog1.FileName <> "" Then
            FilenameBox.Text = LogFileDialog1.FileName
        End If
        Application.DoEvents()

    End Sub



    Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ExitButton.Click
        'Exit Button has been clicked, shut down

        'This should already be the case, but to be sure set the Exit flag
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = True
        End SyncLock
        Me.Hide()

    End Sub



    Private Sub OKButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles OKButton.Click
        'OK Button has been clicked, do final checks and continue

        Dim CurrentSender As New System.Object
        Dim SecondArg As New
System.Windows.Forms.FormClosingEventArgs(CloseReason.None, False)

        'Set the Exit flag to False to continue execution
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = False
        End SyncLock
        StartupForm_FormClosing(CurrentSender, SecondArg)

        Me.Hide()

    End Sub
```

```
    Private Sub StartupForm_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        'Stuff to run when the form is closing
        'loads the form's data into the main class

        SyncLock CentralClass.FileOpsLock
            'set the filename
            CentralClass.Filename = FilenameBox.Text
            'set the append/replace flag
            If RadioButtonAppend.Checked = True Then
                CentralClass.AppendFile = True
            ElseIf RadioButtonReplace.Checked = True Then
                CentralClass.AppendFile = False
            End If
            'set the port speed
            CentralClass.ComPortBaudRate =
CInt((Val(PortSpeedComboBox.SelectedItem)))
            'set the ports
            CentralClass.LaserPort = LaserPortComboBox.SelectedItem.ToString
            CentralClass.SpeedPort =
ControllerPortComboBox.SelectedItem.ToString
            'set the timing feature
            CentralClass.AdvancedTiming = TimingCheckBox.Checked
        End SyncLock

    End Sub



    Private Sub StartupForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Startup events for when the form is loaded

        'Default Selections in the combobox
        ControllerPortComboBox.SelectedIndex = 0
        LaserPortComboBox.SelectedIndex = 0
        PortSpeedComboBox.SelectedIndex = 0
        'default file and location
        FilenameBox.Text =
Environment.GetFolderPath(Environment.SpecialFolder.Desktop) & "\output.txt"

    End Sub



    Private Sub ProgramHelpButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ProgramHelpButton.Click
        'The User wants help
        'HAHAHAHAHAHAHAHAHA

        'MsgBox("Yeah, right.")
        HelpBox.Show()

    End Sub
```

```
    Private Sub StartupForm_Shown(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Shown
        'Runs every time the form is shown, not just the first time it is
loaded

        'Set the Exit flag so that if the form unexpectedly closes the program
ends
        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = True
        End SyncLock

    End Sub



End Class
```

B.17 STEVEDORE.VB

```
Option Explicit On
Option Strict On

Friend NotInheritable Class Stevedore

    'This class handles the primary serial port
    'It controls the speed
    'It also generates the signals for laser control
    'If the mode is correct, those signals will be sent to the primary serial
port
    'However, if two-port mode is being used, the signals will be sent
    'to the other port handler and will be dealt with there
    '
    '
    Friend Shared Col_Sanders As New Thread(AddressOf Stevedore.Colonel)
    '
    '
    'Serial Port 1 (Speed Port, possibly with Laser Control)
    Friend Shared WithEvents COMPort1 As New System.IO.Ports.SerialPort
    Friend Shared ClearToFireLaser As Boolean = False
    Friend Shared SpeedPortLock As New Object
    '
    '
    'Speed Control Port Reader Internal Statics
    Private Shared LastRXTimeTicksSaved As Long = 0
    Private Shared TempRXString As String = ""
    Private Shared NewlineRX As Boolean = False
    Private Shared MainPortReaderLock As New Object
    '
    '
    'Speed Control Port Writer Internal Statics
    Private Shared SavedTXString As String = ""
    Private Shared SavedWriteNumber As Integer = 0
    Private Shared MainPortWriterLock As New Object



    Private Shared Sub Colonel()
```

```
        'This is the single most important part of the program
        'It handles the speed controller serial port
        'It maintains the correct speed depending on program state
        'Issues the laser fire trigger
        'Determines the speed
        'And eats up CPU cycles like a fat guy on weed
        '
        'Individual pins on the serial port should be connected as follows:
        '
        'Laser Fire Signal:  RTS
        'Laser Fire Ground:  GND
        'Speed Controller Signal:  DTR
        'Speed Controller Ground:  GND
        'Speed Sensor Signal:  DSR
        'Speed Sensor Ground:  GND
        'Cavitation Sensor Signal:  CD
        'Cavitation Sensor Ground:  GND
        '
        'If the Laser Data Port is connected to the Speed Control Port (Program
Mode 1)
        'Connect the additional pins as follows:
        '
        'Data TO Laser Signal:  TXD
        'Data TO Laser Ground:  GND
        'Data FROM Laser Signal:  RXD
        'Data FROM Laser Ground:  GND
        '
        'For the Control Pins (NOT TXD, RXD, GND):
        'A POSITIVE Voltage is shown as the Boolean 'TRUE' in the SerialPort
Holding and Enable values
        'A NEGATIVE Voltage is, likewise, a Boolean 'FALSE'
        'The hardware default should be a NEGATIVE voltage on every pin
        '
        'The Advanced Timing Feature is a trick to try to get a more accurate
measurement of the speed
        'The speed is measured by counting changes in a control line on the
serial port
        'However, if the OS preempts the measurement for a long period, some of
those changes might be missed
        'And the speed would therefore get an artificially low measurement
        'So, enabling the Advanced Timing Feature looks for overly large chunks
of missing time
        'The chunks must be more than twice the average loop duration, and
larger than the period for a half-cycle for the speed
        'If it finds such a timeperiod, it will remove that period from the
speed calculations (and consider it a sort of 'dead time')
        'It will replace that long time with the time it takes for an average
Main Loop to execute



        'Declarations
        Dim LocalExecutionStage As Long = 0
        Dim ContinueExecution As Boolean = True
        Dim LocalTXString As String = ""
        Dim LocalRXString As String = ""
        Dim LastTime As Long = 0
        Dim Tock As Long = 0
        Dim Freq As Long = 0
```

```
        Dim LastTimeCheckedRead As Long = 0
        Dim LastTimeCheckedWrite As Long = 0
        Dim LastTimeUpdated As Long = 0
        Dim SendData As Boolean = False
        Dim TimeBetweenReads As Long = 0
        Dim TimeBetweenWriteChecks As Long = 0
        Dim TimeBetweenUpdates As Long = 0
        Dim Proceed As Boolean = False
        Dim LocalPortName As String = ""
        Dim LocalPortSpeed As Integer = 0
        Dim LocalPortDataBits As Integer = 0
        Dim LocalPortStopBits As System.IO.Ports.StopBits =
System.IO.Ports.StopBits.One
        Dim LocalPortFlowControl As System.IO.Ports.Handshake =
System.IO.Ports.Handshake.None
        Dim LocalPortEncoding As System.Text.Encoding =
System.Text.Encoding.UTF8
        Dim LocalPortOpen As Boolean = False
        Dim PortUseString As String = ""
        Dim LocalProgramMode As Long = 0
        Dim SpeedPulsesUp As Double = 0
        Dim SpeedPulsesDown As Double = 0
        Dim CalculatedSpeed As Double = 0
        Dim TargetSpeed As Double = 0
        Dim SpeedError As Double = 0.02
        Dim TargetSpeedAcquired As Boolean = False
        Dim TargetSpeedAcquiredDelayTime As Long = 0
        Dim TargetSpeedAcquiredCounterTocks As Long = 0
        Dim CavitationDetected As Boolean = False
        Dim CavitationDetectedPrevious As Boolean = False
        Dim LocalCavitationAction As Long = 1
        Dim FiredLaser As Boolean = False
        Dim NextLaserPower As Double = 0
        Dim SpeedRestartTimeLeft As Double = 0
        Dim SpeedRestartStatus As Long = 0
        Dim AutoPulsing As Boolean = False
        Dim StopAutopulsing As Boolean = False
        Dim TimeBetweenPulses As Double = 100
        Dim PulseTimer As Long = 0
        Dim StopTimeoutTicks As Long = 100000000
        Dim TimeoutBeginTicks As Long = 0
        Dim CavitationPinPositive As Boolean = False
        Dim CavitationPinPositivePrevious As Boolean = False
        Dim CavitationStartTime As Long = 0
        Dim CavitationTimeoutTicks As Long = 1000000
        Dim CavitationTimeoutTocks As Long = 0
        Dim SpeedDetectorPinPositive As Boolean = False
        Dim SpeedDetectorPinPositivePrevious As Boolean = False
        Dim HalfLifeTicks As Long = 2500000
        Dim DecayConstant As Double = -0.1
        Dim SpeedLambda As Double = 0
        Dim RemainingFraction As Double = 1
        Dim TotalTimeTocks As Double = 0
        Dim MeasuredOutputPinPositiveFraction As Double = 0
        Dim PositiveTimeTocks As Double = 0
        Dim SpeedControlPinPositive As Boolean = False
        Dim LaserFirePinPositive As Boolean = False
        Dim LocalRunButtonClicked As Boolean = False
        Dim LocalRestartDelayTime As Double = 1
```

```
        Dim FormerRestartStatus As Long = 0
        Dim DesiredSpeedPinPositiveFraction As Double = 0
        Dim StopSpeedValue As Double = 0.1
        Dim LastRecordedSpeed As Double = -100
        Dim LastRecordedSpeedTime As Long = 0
        Dim MaxTimeBetweenSpeedRecordingTicks As Long = 100000000
        Dim MaxDeltaSpeedRecording As Double = 5
        Dim SpeedDeltaFraction As Double = 0
        Dim LaserFireStage As Integer = 0
        Dim LocalFireManualPulse As Boolean = False
        Dim NextLaserString As String = ""
        Dim LaserCommandClearanceTimeTocks As Long = 0
        Dim LocalPulseDuration As Double = 0
        Dim LastLaserPulseTimeTocks As Long = 0
        Dim FireNumber As Long = 0
        Dim AutoPulseWaitCounter As Long = 0
        Dim LaserPowerLimit As Double = 100
        Dim LaserSequenceTimeout As Long = 120
        Dim PulseSequenceTimer As Long = 0
        Dim SpinnerStopTimeoutCounter As Long = 0
        Dim LaserEOL As String = ""
        Dim PowerCommand As String = "CAL="
        Dim CommandSeparation As Long = 3500000
        Dim CommandSeparationTimer As Long = 0
        Dim PulseReps As Integer = 0
        Dim MaxPulseReps As Integer = 3
        Dim LaserInitComplete As Boolean = True

        Dim AdvancedTimingFeature As Boolean = False
        Dim UsedAdvancedTiming As Boolean = False
        Dim LoopStartTocks As Long = 0
        Dim LoopEndTocks As Long = 0
        Dim DeltaTocks As Long = 0
        Dim AdvancedDeltaTocks As Long = 0
        Dim LiveTocks As Long = 0
        Dim AllTocks As Long = 0
        Dim RecentTocks As Double = 0
        Dim RecentLoops As Double = 0
        Dim RecentDecayFactor As Double = 0
        Dim MaxDetectableSpeedHalfLife As Double = 1

        Dim RemainingLiveFraction As Double = 0
        Dim LoggedLiveTocks As Long = 0
        Dim LoggedTotalTocks As Long = 0
        Dim RestartDelayTimerTocks As Long = 0
        Dim SpeedZeroCutoff As Double = 0.001

        Dim TotalLoops As Long = 0
        Dim AdvancedTimingCounts As Long = 0

        Dim SpeedDeltaCoefficient As Double = 0.0025 'change rate, pin positive
fraction * seconds / rotation - change
        Dim SpeedDeltaLimiter As Double = 2.0 'max change rate, pin positive
fraction / second

        Dim Look_Ahead_Time As Double = 0.5
        Dim LookAheadTocks As Long
        Dim Regression_Delta As Double
        Dim Regression_xbar As Double
```

```
        Dim Regression_ybar As Double
        Dim Regression_s_xx As Double
        Dim Regression_s_xy As Double
        Dim Regression_s_yy As Double
        Dim Regression_Beta0 As Double
        Dim Regression_Beta1 As Double
        Dim Regression_Counter As Integer = 0
        Dim Regression_nmax As Integer = 5
        Dim Regression_x_Tocks() As Long
        Dim Regression_x() As Double
        Dim Regression_y() As Double

        Dim PredictedSpeed As Double = 0
        Dim DeltaSpeed As Double = 0
        Dim CrossoverSpeed As Double = 8

        Dim JumpCount As Long = 0
        Dim JumpSlope As Double = 2500
        Dim JumpSlopeTock As Double = 0
        Dim LastJumpTock As Long = 0
        Dim TimeBetweenJumps As Double = 1.0
        Dim TocksBetweenJumps As Long = 0
        Dim JumpValue As Double = 0.0000025

        Dim RC As Double = 0    'Resistor x Capacitor, 1/s, set later on
        Dim Capacitor As Double = 0
        Dim PinDeviation As Double = 0
        Dim SpeedOverride As Boolean = False
        Dim FastResume As Double = 0.75 'the fraction of the initial Capacitor
value to use when resuming in pulse sequences



        Try

            'Startup Routine
            Thread.CurrentThread.Name = "Kernel_Sanders"
            'get the program/port mode
            SyncLock Longshoreman.LaserPortLock
                LocalProgramMode = Longshoreman.LaserControlMode
            End SyncLock

            'set the port use string
            If LocalProgramMode = 1 Then
                'shared laser and speed control port
                PortUseString = "Speed Controller/Laser Port"
            Else
                'port only used for speed control
                PortUseString = "Speed Controller Port"
            End If

            'get advanced timing features and laser EOL
            SyncLock CentralClass.FileOpsLock
                AdvancedTimingFeature = CentralClass.AdvancedTiming
                LaserEOL = CentralClass.LaserNewlineString
            End SyncLock

            'set variable defaults and check for the performance counter
            ContinueExecution = True
```

```
            If QueryPerformanceCounter(Tock) = 0 Then
                ContinueExecution = False
                MsgBox("Find a computer with a performance timer.")
                'No performance counter, can't run
                'log the result and set the shutdown flag
                Lumberjack.SendToLog("Computer does not have a performance
timer.  Execution cannot continue, shutting down the Program")
                SyncLock CentralClass.ProgramExitLock
                    CentralClass.ExitProgram = True
                End SyncLock
            End If
            'get the performance counter's frequency (per second)
            QueryPerformanceFrequency(Freq)

            'Set the Target Speed Acquired Countdown, default is one second in
tocks
            TargetSpeedAcquiredDelayTime = Freq * (1)

            'set early values
            If ContinueExecution = True Then
                QueryPerformanceCounter(LastTimeCheckedRead)
                QueryPerformanceCounter(LastTimeCheckedWrite)
                'set the RC-time constant to match the experimental setup
                RC = 150000 * (0.0000047 + 0.00001)
                'Capacitor expresses the stored fraction from 0 to 1 estimated
to exist in the electronics
                Capacitor = 0
                'set time between reads to 1/50 second
                TimeBetweenReads = CLng(Freq / 50)
                'set update frequency to 30/sec
                TimeBetweenUpdates = CLng(Freq / 30)
                'set write check frequency to 60/sec
                TimeBetweenWriteChecks = CLng(Freq / 60)
                'set the decay value in units of 1/tocks (Tocks, NOT Ticks; 1
Tock=1/Freq seconds)
                DecayConstant = -(Math.Log(2) / (HalfLifeTicks * Freq /
10000000))
                'set the conversion factor
                'the divide by two is because it will be used to average the
speed from two different sources
                'the sources are summed up then multiplied by the conversion
factor
                SpeedLambda = (-Freq) * DecayConstant / 2
                'the speed is determined by taking the rate of pulses coming in
and 'smearing' them out
                'using an exponential decay scheme
                'set the cavitation detection timeout period
                CavitationTimeoutTocks = CLng(CavitationTimeoutTicks * Freq /
10000000)
                'amount of time to look ahead and predict the speed
                LookAheadTocks = CLng(Look_Ahead_Time * Freq)
                'set the critical jump slope
                JumpSlopeTock = JumpSlope / Freq
                'set the minimum time between jumps
                TocksBetweenJumps = CLng(TimeBetweenJumps * Freq)
                'determine the delta for use in the Deming Regression
computation
                'Deming Regression is a linear least-squares technique to fit
data to a line and uses the following relationships:
```

```
                '
                ' y ~ b0 + b1 * x
                '
                ' xbar = (1/n) * SUM{ x_i }
                '
                ' ybar = (1/n) * SUM{ y_i }
                '
                ' s_xx = (1/[n-1]) * SUM{ ( x_i - xbar )^2 }
                '
                ' s_yy = (1/[n-1]) * SUM{ ( y_i - ybar )^2 }
                '
                ' s_xy = (1/[n-1]) * SUM{ ( x_i - xbar ) * ( y_i - ybar ) }
                '
                ' d = sigma_y^2 / sigma_x^2    --> delta uses the error in the
x and y samples themselves, not the variances of the sets of x and y points
                '
                ' beta1 = ( s_yy - [d * s_xx] + SQRT{ (s_yy - [d * s_xx])^2 +
(4 * d * s_xy^2) } ) / ( 2 * s_xy )
                ' which is a solution to
                ' -s_xy * beta1^2 + (s_yy - d * s_xx) * beta1 + d * s_xy = 0
                '
                ' beta0 = ybar - (beta1 * xbar)
                '

                'for now, use delta=1 (orthogonal regression) (delta is d
above)
                Regression_Delta = 1
                'the delta is defined as d = variance_y / variance_x
                'where the variance is based on the error in the sample itself
                'using the error propagation formula, the delta here would be
expressed as
                '
                ' d = Speed^2 * DecayConstant^2 * exp[2 * DecayConstant * {Avg.
Loop Tocks}] / ( 1 - exp[2 * DecayConstant * {Avg. Loop Tocks}] )
                '
                'which is not only dependent on speed, but also tends to
produce results on the order of 2E-12
                'if delta --> infinity, then the Deming Regression becomes
standard OLS regression and beta = cov(x,y) / var(x)
                'where the variance and covariance are determined from the
recorded data set undergoing regression analysis
                'if delta --> 0, then beta = var(y) / cov(x,y)
                'An earlier attempt used a dummy loop to estimate loop time and
a speed of 1, and that earlier code is commented out below:
                '
                '
                '
                'QueryPerformanceCounter(Tock)
                'LastTime = Tock
                ''do a few things in a loop to simulate the real loop; the next
few lines through the Do loop are deliberate junk
                'RecentDecayFactor = -10
                'TotalLoops = 0
                'Do
                '    TotalLoops += 1
                '    SyncLock Stevedore.SpeedPortLock
                '        'just because it eats up a few cycles;
                '        RecentDecayFactor = RecentDecayFactor * Math.Exp(-3 *
Math.Log(4) + Math.Log(1.2)) + 3.14
```

```
                '    End SyncLock
                '    QueryPerformanceCounter(Tock)
                '    Application.DoEvents()
                'Loop While TotalLoops < 3
                'TotalLoops = 0
                'If Tock - LastTime <> 0 Then
                '    Regression_Delta = (DecayConstant ^ 2) * Math.Exp(2 *
DecayConstant * (Tock - LastTime)) / (1 - Math.Exp(2 * DecayConstant * (Tock -
LastTime)))
                '    Lumberjack.SendToLog("Using a Deming Variance Ratio of " &
Regression_Delta.ToString & "and a loop time estimated at " & Str(Tock -
LastTime) & " Tocks")
                'Else
                '    'something won't work, exit
                '    Lumberjack.SendToLog("Something Went Wrong with the Deming
Variance Ratio Calculation")
                '    SyncLock CentralClass.ProgramExitLock
                '        CentralClass.ExitProgram = True
                '    End SyncLock
                'End If

            End If

            'set the array sizes for the Deming Regression routine
            ReDim Regression_x_Tocks(Regression_nmax)
            ReDim Regression_x(Regression_nmax)
            ReDim Regression_y(Regression_nmax)
            For Regression_Counter = 0 To Regression_nmax
                Regression_x_Tocks(Regression_Counter) = 0
                Regression_x(Regression_Counter) = 0
                Regression_y(Regression_Counter) = 0
            Next

            'wait for the right time
            Proceed = False
            Do
                'wait for the runbox to finish loading
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.RunBoxReady = True Then
                        'ready to proceed
                        Proceed = True
                    End If
                End SyncLock
                'check for an exit program signal
                SyncLock CentralClass.ProgramExitLock
                    If CentralClass.ExitProgram = True Then
                        'exiting program
                        Proceed = True
                        ContinueExecution = False
                    End If
                End SyncLock
                Thread.Sleep(55)
                Application.DoEvents()
            Loop While Proceed = False

            Thread.Sleep(200)
            Application.DoEvents()
            'then open the serial port
            'get port settings and try to open the port
```

```
            SyncLock CentralClass.FileOpsLock
                LocalPortName = CentralClass.SpeedPort
                LocalPortSpeed = CentralClass.ComPortBaudRate
                LocalPortDataBits = CentralClass.ComPortDataBits
                LocalPortStopBits = CentralClass.ComPortStopBits
                LocalPortFlowControl = CentralClass.ComPortFlowControl
                LocalPortEncoding = CentralClass.ComPortEncoding
            End SyncLock

            Proceed = False
            'open the port
            Do

                Try
                    'log it
                    Lumberjack.SendToLog("Attempting to open the " &
PortUseString & " (" & LocalPortName & ") with the following settings:  " &
LocalPortSpeed.ToString & " baud, " & LocalPortDataBits.ToString & " data bits,
" & LocalPortStopBits.ToString & " stop bits, flow control=" &
LocalPortFlowControl.ToString & ", and " & LocalPortEncoding.ToString & "
Encoding.")
                    'set up and open the port
                    SyncLock Stevedore.SpeedPortLock
                        Stevedore.COMPort1.PortName = LocalPortName
                        Stevedore.COMPort1.BaudRate = LocalPortSpeed
                        Stevedore.COMPort1.DataBits = LocalPortDataBits
                        Stevedore.COMPort1.StopBits = LocalPortStopBits
                        Stevedore.COMPort1.Handshake = LocalPortFlowControl
                        Stevedore.COMPort1.Encoding = LocalPortEncoding
                        'set the read and write buffer size in bytes
                        Stevedore.COMPort1.ReadBufferSize = 16384
                        Stevedore.COMPort1.WriteBufferSize = 16384
                        'open the port
                        Stevedore.COMPort1.Open()
                        LocalPortOpen = Stevedore.COMPort1.IsOpen
                    End SyncLock
                    'log the results
                    If LocalPortOpen = True Then
                        'success
                        Lumberjack.SendToLog("Success in Opening the " &
PortUseString & " (" & LocalPortName & ")")
                        'set the laser pulse and speed pins low
                        SyncLock Stevedore.SpeedPortLock
                            Stevedore.COMPort1.DtrEnable = False
                            Stevedore.COMPort1.RtsEnable = False
                        End SyncLock
                        'proceed with the program
                        Proceed = True
                    Else
                        'some failure without an exception getting caught
                        Lumberjack.SendToLog("Failure in Opening the  " &
PortUseString & " (" & LocalPortName & ")")
                        'see if user wants to try again
                        If MsgBox("Failure in attempt to open the  " &
PortUseString & " (" & LocalPortName & ")" & vbNewLine & "Retry?",
MsgBoxStyle.YesNo, "Port Problem") = MsgBoxResult.Yes Then
                            'user wants to retry
                            Lumberjack.SendToLog("Will retry opening the " &
PortUseString)
```

```
                        Else
                            'user wants to abort
                            Proceed = True
                            Lumberjack.SendToLog("Abort opening " &
PortUseString & ", ending program")
                            SyncLock CentralClass.ProgramExitLock
                                CentralClass.ExitProgram = True
                            End SyncLock
                        End If
                    End If

                Catch SerialException As Exception
                    'it didn't go
                    Lumberjack.SendToLog("Exception in attempt to open the " &
PortUseString & " (" & LocalPortName & ")" & ":  " & vbNewLine &
SerialException.Message)
                    If MsgBox("Exception in attempt to open the " &
PortUseString & " (" & LocalPortName & ")" & ":  " & vbNewLine &
SerialException.Message & vbNewLine & "Retry?", MsgBoxStyle.YesNo, "Laser Port
Problem") = MsgBoxResult.Yes Then
                        'user wants to retry
                        Lumberjack.SendToLog("Will retry opening the " &
PortUseString)
                    Else
                        'user wants to abort
                        Proceed = True
                        Lumberjack.SendToLog("Abort opening " & PortUseString &
", ending program")
                        SyncLock CentralClass.ProgramExitLock
                            CentralClass.ExitProgram = True
                        End SyncLock
                    End If

                End Try

                'Check to see if the program should exit
                SyncLock CentralClass.ProgramExitLock
                    If CentralClass.ExitProgram = True Then
                        Proceed = True
                        ContinueExecution = False
                    End If
                End SyncLock
            Loop While Proceed = False

            'signal that the program is ready
            SyncLock Windowsill.RunBoxLock
                If ContinueExecution = True Then
                    Windowsill.SpeedControlReady = True
                    'signal available laser port if in the correct mode
                    If (LocalProgramMode = 0) Or (LocalProgramMode = 1) Then
                        'in a mode for full control by this thread
                        Windowsill.LaserPortReady = True
                    End If
                End If
            End SyncLock


            'Main Loop follows
            'It is the whole point of the program
```

```
            'check for an exit condition
            SyncLock CentralClass.ProgramExitLock
                ContinueExecution = Not CentralClass.ExitProgram
            End SyncLock

            If ContinueExecution = True Then
                'start of main loop in tocks
                QueryPerformanceCounter(LoopStartTocks)
            End If

            Do While (ContinueExecution = True)

                'This is the main loop of the program
                'It executes continuously while the program is in execution
mode; it does not sleep
                'It monitors the speed and cavitation
                'It controls the speed and laser pulses
                'It handles RXD on the main serial port
                'And TXD as well if the laser is on it

                TotalLoops += 1

                'Clock (using Win32 Performance Counters)
                LastTime = Tock
                QueryPerformanceCounter(Tock)
                DeltaTocks = Tock - LastTime

                'advanced timing calculations here
                If AdvancedTimingFeature = True Then
                    'calculate advanced timing if necessary
                    If CalculatedSpeed > (1.0R / CDbl(Freq)) Then
                        'nonzero speed, can proceed
                        'using 1/freq above is to prevent overflows
                        If (DeltaTocks > (2L * (Tock - LoopStartTocks) /
TotalLoops)) And (DeltaTocks > CLng((0.75R) * Freq / CalculatedSpeed)) Then
                            'only use advanced calculation if there appears to
be a chunk of time missing; i.e., the OS took over for a bit
                            'and only if it might have wiped out speed pulses;
                            'the missing chunk of time must be greater than
0.75 times the pulse period
                            'and must also be greater than twice the average
loop time
                            '(a pulse can be wiped out at ~1/2 of the pulse
period, but is somewhat unlikely)
                            'If so, the total elapsed time will be replaced by
the average loop time
                            AdvancedDeltaTocks = CLng((Tock - LoopStartTocks) /
TotalLoops)
                            AdvancedTimingCounts += 1
                            'if not already used, mark the use of advanced
timing
                            If UsedAdvancedTiming = False Then
                                'mark it as true
                                UsedAdvancedTiming = True
                                'log it
                                Lumberjack.SendToLog("Begin Use of Advanced
Timing Features")
                                'start the live time logging variables
```

```
                            LoggedTotalTocks = LastTime
                            LoggedLiveTocks = LiveTocks
                        End If
                    Else
                        AdvancedDeltaTocks = DeltaTocks
                    End If
                Else
                    'speed is zero
                    AdvancedDeltaTocks = DeltaTocks
                End If
            Else
                'disabled advanced timing
                AdvancedDeltaTocks = DeltaTocks
            End If

            'record the live time
            LiveTocks = LiveTocks + AdvancedDeltaTocks
            AllTocks = AllTocks + DeltaTocks

            'Calculations for the max detectable speed
            RecentDecayFactor = Math.Exp(-Math.Log(2) * AdvancedDeltaTocks
/ (Freq * MaxDetectableSpeedHalfLife))
            RecentLoops = (RecentLoops * RecentDecayFactor) + 1
            RecentTocks = (RecentTocks * RecentDecayFactor) +
AdvancedDeltaTocks

            'Put Input Event handlers here
            'check the speed and cavitation sensor pins
            'as well as the 'Fire Laser' and Speed Controller pins
            'store the previous values
            SpeedDetectorPinPositivePrevious = SpeedDetectorPinPositive
            CavitationPinPositivePrevious = CavitationPinPositive
            SyncLock Stevedore.SpeedPortLock
                If Stevedore.COMPort1.IsOpen = True Then
                    'get the speed pin status
                    SpeedDetectorPinPositive =
Stevedore.COMPort1.DsrHolding
                    'get the cavitation pin status
                    'positive means the instrument can receive greater
amounts of light than the negative status
                    'which should happen when a vapor column appears
                    CavitationPinPositive = Stevedore.COMPort1.CDHolding
                    'get the speed controller pin status
                    SpeedControlPinPositive = Stevedore.COMPort1.DtrEnable
                    'get the laser fire pin status
                    LaserFirePinPositive = Stevedore.COMPort1.RtsEnable
                Else
                    'the port is not open, close the program
                    ContinueExecution = False
                End If
            End SyncLock
            If ContinueExecution = False Then
                'signal a program exit
                SyncLock CentralClass.ProgramExitLock
                    CentralClass.ExitProgram = True
                End SyncLock
            End If

            'process the input pin status meanings
```

```
                'if the speed pin's current state and previous state are
different, increment the appropriate counter
                If SpeedDetectorPinPositive <> SpeedDetectorPinPositivePrevious
Then
                    'speed pulse received, see if pin went high
                    If SpeedDetectorPinPositive = True Then
                        'the pin went positive
                        'increment the speedpulsesup counter
                        SpeedPulsesUp += 1
                    Else
                        'the pin went negative
                        'increment the other counter
                        SpeedPulsesDown += 1
                    End If
                End If
                'do some speed processing:  calculate the remaining 'sum' of
pulses
                'it uses the advanced timing variable, but if advanced timing
is disabled, the advanced variable should equal the standard one
                RemainingFraction = Math.Exp(DecayConstant * CDbl(DeltaTocks))
                RemainingLiveFraction = Math.Exp(DecayConstant *
CDbl(AdvancedDeltaTocks))
                SpeedPulsesUp = SpeedPulsesUp * RemainingLiveFraction
                SpeedPulsesDown = SpeedPulsesDown * RemainingLiveFraction

                'calculate timing
                'calculate the total time
                TotalTimeTocks = (TotalTimeTocks + CDbl(DeltaTocks)) *
RemainingFraction
                If SpeedOverride = False Then
                    'The speed isn't overridden, behave normally
                    'LiveTimeTocks = (LiveTimeTocks + CDbl(AdvancedDeltaTocks))
* RemainingLiveFraction
                    'calculate the time the output pin is positive and update
the Capacitor estimate
                    If SpeedControlPinPositive = True Then
                        'pin is positive, add to the time
                        PositiveTimeTocks = (PositiveTimeTocks +
CDbl(DeltaTocks)) * RemainingFraction
                        'bring the capacitor fraction closer to 1
                        Capacitor = 1 - (1 - Capacitor) * Math.Exp(-
(CDbl(DeltaTocks) / Freq) / RC)
                    Else
                        'pin is negative, lose time
                        PositiveTimeTocks = PositiveTimeTocks *
RemainingFraction
                        'decay the capacitor fraction toward zero
                        Capacitor = Capacitor * Math.Exp(-(CDbl(DeltaTocks) /
Freq) / RC)
                    End If
                    'calculate the fraction of the time that the speed control
output pin is positive
                    If TotalTimeTocks > 0 Then
                        'avoiding a divide-by-zero error
                        MeasuredOutputPinPositiveFraction = PositiveTimeTocks /
TotalTimeTocks
                    Else
                        'total time is zero (unlikely to be negative)
                        MeasuredOutputPinPositiveFraction = 0
```

```
                    End If
                Else
                    'the speed is overridden
                    'this is off-normal operation
                    'will not adjust the capacitor value here, allowing fast
resumption after the off-normal condition ends
                    'sed the measured output value to the desired value to
avoid extra deviation behavior
                    MeasuredOutputPinPositiveFraction =
DesiredSpeedPinPositiveFraction
                End If

                'process the cavitation status
                CavitationDetectedPrevious = CavitationDetected
                If CavitationPinPositive = True Then
                    'cavitation detected?  give it a timeout to make sure
                    If CavitationPinPositive <> CavitationPinPositivePrevious
Then
                        'new signal, start the countdown
                        CavitationStartTime = Tock
                    Else
                        'continuing signal, see if the countdown has completed
                        If Tock > (CavitationStartTime +
CavitationTimeoutTocks) Then
                            'countdown has finished without sensing a non-
cavitation condition in the meantime
                            CavitationDetected = True
                            'log it if it's a first-time event
                            If CavitationDetectedPrevious = False Then
                                'first-time event, log the detection
                                Lumberjack.SendToLog("Cavitation Detected!
Speed is " & CalculatedSpeed.ToString)
                            End If
                        End If
                    End If
                Else
                    'Cavitation not detected
                    CavitationDetected = False
                    'log termination of cavitation condition if it existed
                    If CavitationDetectedPrevious = True Then
                        'cavitation condition just stopped, log it
                        Lumberjack.SendToLog("Cavitation Condition No Longer
Detected")
                    End If
                End If
                If CavitationDetected = True Then
                    'for Mode 1, Press Stop
                    If LocalCavitationAction = 1 Then
                        SyncLock Windowsill.RunBoxLock
                            Windowsill.StopAction = True
                        End SyncLock
                    End If
                    'Terminate Autopulsing on Cavitation Detection
                    If AutoPulsing = True Then
                        AutoPulsing = False
                        StopAutopulsing = True
                    End If
                End If
```

```
                'Output Event handlers go here
                'Largely for changes in serial port control pins
                'Somewhat mischaracterized; Events are unused

                'update speed control output pin
                SyncLock Stevedore.SpeedPortLock
                    'only do it if the port is still open
                    If Stevedore.COMPort1.IsOpen = True Then
                        'set/make sure the pin is in the correct state for the
particular case
                        If (DesiredSpeedPinPositiveFraction = 0) Or
(SpeedOverride = True) Then
                            'it should be set to negative, make sure of it
                            If SpeedControlPinPositive = True Then
                                'it's not set to negative, set it
                                Stevedore.COMPort1.DtrEnable = False
                            End If
                        ElseIf DesiredSpeedPinPositiveFraction = 1 Then
                            'it should be set to positive, make sure
                            If SpeedControlPinPositive = False Then
                                'it's negative, set it positive
                                Stevedore.COMPort1.DtrEnable = True
                            End If
                        Else
                            'see if it's time to change pin states
                            'if the desired fraction is equal to the measured
fraction, leave it alone
                            If DesiredSpeedPinPositiveFraction >
MeasuredOutputPinPositiveFraction Then
                                'it should be set to positive
                                If SpeedControlPinPositive = False Then
                                    'change it
                                    Stevedore.COMPort1.DtrEnable = True
                                End If
                            ElseIf DesiredSpeedPinPositiveFraction <
MeasuredOutputPinPositiveFraction Then
                                'it should be set negative
                                If SpeedControlPinPositive = True Then
                                    'change it
                                    Stevedore.COMPort1.DtrEnable = False
                                End If
                            Else
                                'LEAVE IT ALONE!
                                'That is all
                            End If
                        End If
                    End If
                End SyncLock

                'Fire Control Block
                'This section handles the "Fire Laser!" pin
                'and associated Laser Logic
                'The firing logic goes through a number of stages
                'only one stage executes in a given progression through the
block; once per major loop in this thread
                'these stages control various parts of the firing sequence:
taking over the laser lines, adjusting the power, timers, issues pulses, etc.
                'It gets a bit messy
```

```
                SpeedOverride = False
                If LocalProgramMode >= 1 Then
                    'Laser Control Mode is enabled
                    SyncLock Windowsill.RunBoxLock
                        LaserInitComplete = Windowsill.FinishedStartup
                    End SyncLock
                    If LocalFireManualPulse = True Then
                        'the user wants to fire a pulse, follow that sequence
                        'clear autopulsing values
                        If AutoPulsing = True Then
                            AutoPulsing = False
                            StopAutopulsing = True
                        End If
                        If LaserFireStage = 0 Then
                            PulseReps = 0
                            'go to stage 100
                            LaserFireStage = 100
                            'start the timer
                            AutoPulseWaitCounter = Tock
                            PulseSequenceTimer = Tock
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                        ElseIf (LaserFireStage > 0) And (LaserFireStage < 100)
Then
                            'an autopulse was interrupted, go to stage 0 up
before proceeding
                            'log it
                            Lumberjack.SendToLog("A Laser Pulse Operation
appears to have been interrupted at Stage " & LaserFireStage.ToString)
                            LaserFireStage = 0
                        ElseIf LaserFireStage = 100 Then
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            'proceed once stopped
                            If CalculatedSpeed < StopSpeedValue Then
                                LaserFireStage = 150
                            End If
                            'check the timeout (120 seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 150 Then
                            'lock down the laser dataline and transmit the
laser pulse power
                            'form the command that will adjust the laser power
                            'it should use NextLaserPower, which is copied from
windowsill.nextlaserpulsepower
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            If NextLaserPower > LaserPowerLimit Then
                                'too high, cancel pulse
                                NextLaserPower = 1
```

```
                                'log the event
                                Lumberjack.SendToLog("Pulse Power Has Exceeded
Limits:  Cancel Manual Pulse")
                                'cancel the pulse
                                SyncLock Windowsill.RunBoxLock
                                    Windowsill.FireManualPulse = False
                                End SyncLock
                                'end the sequence
                                LocalFireManualPulse = False
                                LaserFireStage = 0
                            Else
                                NextLaserString = LaserEOL
                                'log the transmission
                                Lumberjack.SendToLog("Automatically Generated
Command to be sent to the laser:  " & NextLaserString)
                                SyncLock Windowsill.RunBoxLock
                                    'lock down the dataline
                                    Windowsill.LaserTXDataCritical = True
                                    'send the next power command to the laser
head
                                    Windowsill.CommandToLaser =
Windowsill.CommandToLaser & NextLaserString
                                End SyncLock
                                'proceed to the next step
                                NextLaserString = ""
                                LaserFireStage = 200
                                CommandSeparationTimer = Now.Ticks
                            End If
                            'check the timeout (x seconds since beginning
stage)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 200 Then
                            'wait for the bufferred command to complete its
transmission
                            'see if the shared buffer has cleared
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            SyncLock Windowsill.RunBoxLock
                                NextLaserString = Windowsill.CommandToLaser
                            End SyncLock
                            If NextLaserString = "" Then
                                'it's been loaded past the shared buffer, see
if the system is ready
                                SyncLock Stevedore.SpeedPortLock
                                    If Stevedore.ClearToFireLaser = True Then
                                        If Now.Ticks - CommandSeparationTimer >
CommandSeparation Then
                                            'proceed to the next stage
                                            NextLaserString = PowerCommand &
NextLaserPower.ToString & LaserEOL
                                            'log the transmission
```

```
                                            Lumberjack.SendToLog("Automatically
Generated Command to be sent to the laser:  " & NextLaserString)
                                            SyncLock Windowsill.RunBoxLock
                                                'lock down the dataline
                                                Windowsill.LaserTXDataCritical
= True
                                                'send the next power command to
the laser head
                                                Windowsill.CommandToLaser =
Windowsill.CommandToLaser & NextLaserString
                                            End SyncLock
                                            'proceed to the next step
                                            NextLaserString = ""
                                            LaserFireStage = 250
                                            LaserCommandClearanceTimeTocks =
Tock
                                        End If
                                    Else
                                        'update the timer so that the laser has
some time to process the pre-command eol
                                        CommandSeparationTimer = Now.Ticks
                                    End If
                                End SyncLock
                            Else
                                'update the timer so that the laser has some
time to process the pre-command eol
                                CommandSeparationTimer = Now.Ticks
                            End If
                            'check the timeout (120 seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 250 Then
                            'wait for the bufferred command to complete its
transmission
                            'see if the shared buffer has cleared
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            SyncLock Windowsill.RunBoxLock
                                NextLaserString = Windowsill.CommandToLaser
                            End SyncLock
                            If NextLaserString = "" Then
                                'it's been loaded past the shared buffer, see
if the system is ready
                                SyncLock Stevedore.SpeedPortLock
                                    If Stevedore.ClearToFireLaser = True Then
                                        'proceed to the next stage
                                        LaserFireStage = 300
                                        LaserCommandClearanceTimeTocks = Tock
                                    End If
                                End SyncLock
                            End If
                            'check the timeout (120 seconds since beginning)
```

```
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 300 Then
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            'this step is a timer to ensure the laser has
processed the command (4 seconds) fully
                            If (CDbl(Tock - LaserCommandClearanceTimeTocks) /
CDbl(Freq)) > 4 Then
                                'proceed
                                LaserFireStage = 350
                                'compensate for an expected overreaction; the
speed will come back online after this
                                Capacitor = Capacitor * FastResume
                            End If
                            'check the timeout (120 seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 350 Then
                            'start the spinup counter and move to the next
stage
                            LaserFireStage = 400
                            LaserCommandClearanceTimeTocks = Tock
                            'check the timeout (120 seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 400 Then
                            'wait 20 seconds as the speed goes back to where it
was before being overridden
                            If (CDbl(Tock - LaserCommandClearanceTimeTocks) /
CDbl(Freq)) > 20 Then
                                'proceed
                                LaserFireStage = 450
                            End If
                            'check the timeout (x seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
```

```
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 450 Then
                            'start the delay between pulses
                            AutoPulseWaitCounter = Tock
                            'proceed to the next stage
                            LaserFireStage = 500
                        ElseIf LaserFireStage = 500 Then
                            'wait for delay to complete
                            If CDbl(Tock - AutoPulseWaitCounter) / CDbl(Freq) >
TimeBetweenPulses Then
                                'delay has finished, proceed
                                LaserFireStage = 550
                                AutoPulseWaitCounter = Tock
                            End If
                            'check the timeout (x seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
                            End If
                        ElseIf LaserFireStage = 550 Then
                            'FIRE!
                            FireNumber += 1
                            'log the event
                            Lumberjack.SendToLog("Firing Laser with a Pulse
Power of " & NextLaserPower.ToString & " and an estimated duration of " &
LocalPulseDuration.ToString)
                            Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                            Lumberjack.SendToLog("FIRE " & FireNumber.ToString
& "!")
                            'Fire the Laser:  Raise the RTS signal
                            SyncLock Stevedore.SpeedPortLock
                                Stevedore.COMPort1.RtsEnable = True
                            End SyncLock
                            'raise the fired laser flags
                            FiredLaser = True
                            'update the last fire time
                            LastLaserPulseTimeTocks = Tock
                            'proceed to the next stage
                            LaserFireStage = 600
                            'check the timeout (x seconds since beginning)
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel pulsing, and log it
                                Lumberjack.SendToLog("Manual Pulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating Pulse")
                                LaserFireStage = 0
                                LocalFireManualPulse = False
```

```
                            End If
                        ElseIf LaserFireStage = 600 Then
                            'complete the firing sequence
                            'lower the fired laser flag
                            SyncLock Stevedore.SpeedPortLock
                                Stevedore.COMPort1.RtsEnable = False
                            End SyncLock
                            'release the laser dataline
                            SyncLock Windowsill.RunBoxLock
                                Windowsill.LaserTXDataCritical = False
                                Windowsill.FireManualPulse = False
                            End SyncLock
                            'end the sequence
                            LocalFireManualPulse = False
                            LaserFireStage = 0
                        Else
                            'Unknown stage
                            'log the event and clear it
                            Lumberjack.SendToLog("Unknown Fire Control Stage:
Manual Pulse Stage " & LaserFireStage.ToString)
                            LocalFireManualPulse = False
                            LaserFireStage = 0
                        End If

                    ElseIf AutoPulsing = True Then
                        'automatic fire control sequence
                        If LocalRunButtonClicked = False Then
                            'not controlling the speed, do not fire
                            If LaserFireStage <> 0 Then
                                'interrupting a pulse, log it
                                Lumberjack.SendToLog("A Laser Pulse Operation
appears to have been interrupted at Stage " & LaserFireStage.ToString)
                                LaserFireStage = 0
                            End If
                            LaserFireStage = 0
                        ElseIf LaserFireStage = 0 Then
                            'go to the first stage
                            LaserFireStage = 5
                        ElseIf LaserFireStage >= 100 Then
                            'a manual pulse was interrupted, go to stage 0
first
                            'log it
                            Lumberjack.SendToLog("A Laser Pulse Operation
appears to have been interrupted at Stage " & LaserFireStage.ToString)
                            LaserFireStage = 0
                        ElseIf LaserFireStage = 5 Then
                            'get things started and go on
                            'start the timer
                            AutoPulseWaitCounter = Tock
                            PulseSequenceTimer = Tock
                            'go to the next step
                            LaserFireStage = 10
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            'set the number of pulse repetitions to 0
                            PulseReps = 0
                        ElseIf LaserFireStage = 10 Then
                            'check to see if the conditions are right to
proceed
```

```
                            'or if it's timed out
                            If (CavitationDetected = False) And
(CalculatedSpeed < StopSpeedValue) Then
                                'conditions are right to move on to the next
step
                                LaserFireStage = 15
                            End If
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 15 Then
                            'lock down the laser dataline and transmit the
laser pulse power
                            'form the command that will adjust the laser power
                            'it should use NextLaserPower, which is copied from
windowsill.nextlaserpulsepower
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            If NextLaserPower > LaserPowerLimit Then
                                'too high, end autopulsing
                                NextLaserPower = 1
                                StopAutopulsing = True
                                AutoPulsing = False
                                'log the event
                                Lumberjack.SendToLog("Autopulsing Power Has
Exceeded Limits:  End Autopulsing")
                                LaserFireStage = 0
                            Else
                                NextLaserString = LaserEOL
                                'log the transmission
                                Lumberjack.SendToLog("Automatically Generated
Command to be sent to the laser:  " & NextLaserString)
                                SyncLock Windowsill.RunBoxLock
                                    'lock down the dataline
                                    Windowsill.LaserTXDataCritical = True
                                    'send the next power command to the laser
head
                                    Windowsill.CommandToLaser =
Windowsill.CommandToLaser & NextLaserString
                                End SyncLock
                                'proceed to the next step
                                NextLaserString = ""
                                LaserFireStage = 20
                                CommandSeparationTimer = Now.Ticks
                            End If
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
```

```
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 20 Then
                            'wait for the bufferred command to complete its
transmission
                            'see if the shared buffer has cleared
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            SyncLock Windowsill.RunBoxLock
                                NextLaserString = Windowsill.CommandToLaser
                            End SyncLock
                            If NextLaserString = "" Then
                                'it's been loaded past the shared buffer, see
if the system is ready
                                SyncLock Stevedore.SpeedPortLock
                                    If Stevedore.ClearToFireLaser = True Then
                                        'check for a completed delay from the
last command
                                        If Now.Ticks - CommandSeparationTimer >
CommandSeparation Then
                                            'issue the next laser command
                                            'and proceed to the next stage
                                            NextLaserString = PowerCommand &
NextLaserPower.ToString & LaserEOL
                                            'log the transmission
                                            Lumberjack.SendToLog("Automatically
Generated Command to be sent to the laser:  " & NextLaserString)
                                            SyncLock Windowsill.RunBoxLock
                                                'send the next power command to
the laser head
                                                Windowsill.CommandToLaser =
Windowsill.CommandToLaser & NextLaserString
                                            End SyncLock
                                            'proceed to the next step
                                            NextLaserString = ""
                                            LaserFireStage = 25
                                            LaserCommandClearanceTimeTocks =
Tock
                                        End If
                                    Else
                                        'update the timer so that the laser has
some time to process the pre-command eol
                                        CommandSeparationTimer = Now.Ticks
                                    End If
                                End SyncLock
                            Else
                                'update the timer so that the laser has some
time to process the pre-command eol
                                CommandSeparationTimer = Now.Ticks
                            End If
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
```

```
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 25 Then
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            'wait for the bufferred command to complete its
transmission
                            'see if the shared buffer has cleared
                            SyncLock Windowsill.RunBoxLock
                                NextLaserString = Windowsill.CommandToLaser
                            End SyncLock
                            If NextLaserString = "" Then
                                'it's been loaded past the shared buffer, see
if the system is ready
                                SyncLock Stevedore.SpeedPortLock
                                    If Stevedore.ClearToFireLaser = True Then
                                        'proceed to the next stage
                                        LaserFireStage = 30
                                        LaserCommandClearanceTimeTocks = Tock
                                    End If
                                End SyncLock
                            End If
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 30 Then
                            'halt the speed to adjust laser power
                            SpeedOverride = True
                            'this step is just a short timer to ensure the
laser has processed the command (4 seconds)
                            If (CDbl(Tock - LaserCommandClearanceTimeTocks) /
CDbl(Freq)) > 4 Then
                                'proceed
                                LaserFireStage = 35
                                'compensate for an expected overreaction; the
speed will come back online after this
                                Capacitor = Capacitor * FastResume
                            End If
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
```

```
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 35 Then
                            'start the delay between pulses
                            AutoPulseWaitCounter = Tock
                            'proceed to the next stage
                            LaserFireStage = 40
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 40 Then
                            'wait for delay to complete
                            If CDbl(Tock - AutoPulseWaitCounter) / CDbl(Freq) >
TimeBetweenPulses Then
                                'delay has finished, proceed
                                LaserFireStage = 45
                                AutoPulseWaitCounter = Tock
                            End If
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        ElseIf LaserFireStage = 45 Then
                            'FIRE!!!
                            'or something
                            If CavitationDetected = True Then
                                'stop autopulsing on cavitation detection
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                                'unblock the datalink
                                SyncLock Windowsill.RunBoxLock
                                    Windowsill.LaserTXDataCritical = False
                                End SyncLock
                                'log it
                                Lumberjack.SendToLog("Cavitation Detected:  End
Autopulsing")
                            ElseIf (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
```

```
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            ElseIf (TargetSpeedAcquired = True) And
(CavitationPinPositive = False) Then
                                'ready to fire, find the correct theta:
                                If (SpeedDetectorPinPositive = True) And
SpeedDetectorPinPositivePrevious = False Then
                                    'found the correct theta (just after the
detector pin goes positive)
                                    'FIRE LASER!!!!
                                    'FIRE!
                                    FireNumber += 1
                                    PulseReps += 1
                                    'log the event
                                    Lumberjack.SendToLog("Firing Laser with a
Pulse Power of " & NextLaserPower.ToString & " and an estimated duration of " &
LocalPulseDuration.ToString)
                                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                                    Lumberjack.SendToLog("FIRE " &
FireNumber.ToString & "!")
                                    'Fire the Laser:  Raise the RTS signal
                                    SyncLock Stevedore.SpeedPortLock
                                        Stevedore.COMPort1.RtsEnable = True
                                    End SyncLock
                                    'raise the fired laser flags
                                    FiredLaser = True
                                    'update the last fire time
                                    LastLaserPulseTimeTocks = Tock
                                    'proceed to the next stage
                                    LaserFireStage = 50
                                End If
                            End If
                        ElseIf LaserFireStage = 50 Then
                            'complete the firing sequence
                            'lower the fire laser signal
                            SyncLock Stevedore.SpeedPortLock
                                Stevedore.COMPort1.RtsEnable = False
                            End SyncLock
                            'Autopulse mode fires a cluster of pulses with a
common energy, and only shifts the energy value between clusters
                            If PulseReps >= MaxPulseReps Then
                                'the cluster of pulses has completed, move on
                                'release the laser dataline
                                SyncLock Windowsill.RunBoxLock
                                    Windowsill.LaserTXDataCritical = False
                                End SyncLock
                                'update the pulse energy
                                NextLaserPower = NextLaserPower + 1
                                If NextLaserPower > LaserPowerLimit Then
                                    'too high, end autopulsing
                                    NextLaserPower = 1
                                    StopAutopulsing = True
                                    AutoPulsing = False
                                    'log the event
                                    Lumberjack.SendToLog("Autopulsing Power Has
Exceeded Limits:  End Autopulsing")
                                End If
```

```
                                SyncLock Windowsill.RunBoxLock
                                    Windowsill.NextLaserPulsePower =
NextLaserPower
                                End SyncLock
                                'end the sequence
                                LaserFireStage = 55
                                AutoPulseWaitCounter = Tock
                            Else
                                'the cluster of pulses has not completed,
repeat the pulse from the delay between them
                                LaserFireStage = 35
                                'check for a timeout
                                If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                    'timed out (default is 120 seconds)
                                    'cancel autopulsing, and log it
                                    Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                    LaserFireStage = 0
                                    StopAutopulsing = True
                                    AutoPulsing = False
                                End If
                            End If
                        ElseIf LaserFireStage = 55 Then
                            'end the sequence
                            'this is just a post-cluster timer to give the
cavitation a chance to develop before the next sequence is applied
                            'wait for delay to complete
                            If CDbl(Tock - AutoPulseWaitCounter) / CDbl(Freq) >
TimeBetweenPulses Then
                                'delay has finished, so has the sequence
                                LaserFireStage = 0
                                AutoPulseWaitCounter = Tock
                            End If
                            'check for a timeout
                            If (Tock - PulseSequenceTimer) / Freq >
LaserSequenceTimeout Then
                                'timed out (default is 120 seconds)
                                'cancel autopulsing, and log it
                                Lumberjack.SendToLog("AutoPulse Timeout in
Stage " & LaserFireStage.ToString & "; terminating AutoPulse Mode")
                                LaserFireStage = 0
                                StopAutopulsing = True
                                AutoPulsing = False
                            End If
                        Else
                            'Unknown stage
                            'log the event and clear it
                            Lumberjack.SendToLog("Unknown Fire Control Stage:
Autopulse Stage " & LaserFireStage.ToString)
                            LaserFireStage = 0
                        End If
                    Else
                        'fire control disabled
                        If LaserFireStage <> 0 Then
                            'fire control was disabled in firing operations
                            'log the pulse cancellation
                            Lumberjack.SendToLog("Current Laser Pulse Operation
Cancelled at Stage " & LaserFireStage.ToString)
```

```
                        LaserFireStage = 0
                    End If
                End If
            Else
                'fire control disabled
                LaserFireStage = 0
            End If
            If LaserFireStage = 0 Then
                'make sure the laser is not being told to fire at this
point
                SyncLock Stevedore.SpeedPortLock
                    If Stevedore.COMPort1.RtsEnable = True Then
                        'lower the laser fire signal
                        Stevedore.COMPort1.RtsEnable = False
                    End If
                End SyncLock
                'make sure the laser datalink is unblocked
                If LaserInitComplete = True Then
                    SyncLock Windowsill.RunBoxLock
                        Windowsill.LaserTXDataCritical = False
                    End SyncLock
                End If
            End If


            'Timed events go here
            'check for updated speed settings here?
            'update calculated speed
            'etc.

            'update logged speed if the conditions are met
            'those include time and speed change
            If (Tock - LastRecordedSpeedTime) / Freq >
(MaxTimeBetweenSpeedRecordingTicks / 10000000) Then
                'it's time to log the speed
                Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                'update the speed recording time and value
                LastRecordedSpeed = CalculatedSpeed
                LastRecordedSpeedTime = Tock
                'log the live fraction, if applicable
                If UsedAdvancedTiming = True Then
                    'advanced timing has been applied
                    Lumberjack.SendToLog("Current Live Fraction:  " &
(CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock - LoggedTotalTocks)).ToString)
                    'update the logged times
                    LoggedLiveTocks = LiveTocks
                    LoggedTotalTocks = Tock
                End If
            ElseIf CalculatedSpeed > StopSpeedValue Then
                'it is not stopped
                'check for standard delta
                If Math.Abs(CalculatedSpeed - LastRecordedSpeed) >
MaxDeltaSpeedRecording Then
                    'Speed is different enough to update in the log
                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                    'update the speed recording time and value
                    LastRecordedSpeed = CalculatedSpeed
```

```
                        LastRecordedSpeedTime = Tock
                        'log the live fraction, if applicable
                        If UsedAdvancedTiming = True Then
                            'advanced timing has been applied
                            Lumberjack.SendToLog("Current Live Fraction:  " &
(CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock - LoggedTotalTocks)).ToString)
                            'update the logged times
                            LoggedLiveTocks = LiveTocks
                            LoggedTotalTocks = Tock
                        End If
                    ElseIf (TargetSpeed > 0) And (LocalRunButtonClicked = True)
Then
                        'see if it has crossed in to the fine-tune region or if
it changed enough within it
                        If Math.Abs((CalculatedSpeed - TargetSpeed) /
TargetSpeed) <= 1.25 * SpeedError Then
                            'in fine-tune region
                            'see if it crossed into the fine-tune region
                            If Math.Abs((LastRecordedSpeed - TargetSpeed) /
TargetSpeed) > 1.25 * SpeedError Then
                                'it crossed into the fine-tune region, log it
                                Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                                'update the speed recording time and value
                                LastRecordedSpeed = CalculatedSpeed
                                LastRecordedSpeedTime = Tock
                                'log the live fraction, if applicable
                                If UsedAdvancedTiming = True Then
                                    'advanced timing has been applied
                                    Lumberjack.SendToLog("Current Live
Fraction:  " & (CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock -
LoggedTotalTocks)).ToString)
                                    'update the logged times
                                    LoggedLiveTocks = LiveTocks
                                    LoggedTotalTocks = Tock
                                End If
                            ElseIf (Math.Abs((LastRecordedSpeed -
CalculatedSpeed) / TargetSpeed) > (SpeedError / 3)) Or
(Math.Abs(LastRecordedSpeed - CalculatedSpeed) > (MaxDeltaSpeedRecording / 5))
Then
                                'big enough change in the fine-tune region to
log
                                'add one more requirement:  minimum change
                                If Math.Abs(LastRecordedSpeed -
CalculatedSpeed) > 0.5 Then
                                    'this prevents microscopic changes from
being logged
                                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                                    'update the speed recording time and value
                                    LastRecordedSpeed = CalculatedSpeed
                                    LastRecordedSpeedTime = Tock
                                    'log the live fraction, if applicable
                                    If UsedAdvancedTiming = True Then
                                        'advanced timing has been applied
                                        Lumberjack.SendToLog("Current Live
Fraction:  " & (CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock -
LoggedTotalTocks)).ToString)
                                        'update the logged times
```

```
                                    LoggedLiveTocks = LiveTocks
                                    LoggedTotalTocks = Tock
                                End If
                            End If
                        End If
                        'difference in the fine-tune region is big enough
                    ElseIf Math.Abs((LastRecordedSpeed - TargetSpeed) /
TargetSpeed) <= 1.25 * SpeedError Then
                        'moved outside target range
                        'log it
                        Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                        'update the speed recording time and value
                        LastRecordedSpeed = CalculatedSpeed
                        LastRecordedSpeedTime = Tock
                        'log the live fraction, if applicable
                        If UsedAdvancedTiming = True Then
                            'advanced timing has been applied
                            Lumberjack.SendToLog("Current Live Fraction:  "
& (CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock - LoggedTotalTocks)).ToString)
                            'update the logged times
                            LoggedLiveTocks = LiveTocks
                            LoggedTotalTocks = Tock
                        End If
                    End If
                End If
            Else
                'it's currently stopped, see if it was before
                If LastRecordedSpeed > StopSpeedValue Then
                    'wasn't stopped last time, log it
                    Lumberjack.SendToLog("Speed Effectively Stopped")
                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                    'update the speed recording time and value
                    LastRecordedSpeed = CalculatedSpeed
                    LastRecordedSpeedTime = Tock
                    'log the live fraction, if applicable
                    If UsedAdvancedTiming = True Then
                        'advanced timing has been applied
                        Lumberjack.SendToLog("Current Live Fraction:  " &
(CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock - LoggedTotalTocks)).ToString)
                        'update the logged times
                        LoggedLiveTocks = LiveTocks
                        LoggedTotalTocks = Tock
                    End If
                End If
            End If

            'Read the Serial Port data, if it's time
            If Tock > LastTimeCheckedRead + TimeBetweenReads Then
                'check for data in the serial port
                Stevedore.MainPortReader()
                'update checked time
                LastTimeCheckedRead = Tock
            End If

            'Transmit data through the Speed Port (Mode 1 only)
            If LocalProgramMode = 1 Then
                SendData = False
```

```
                    'check for write
                    SyncLock Stevedore.MainPortWriterLock
                        'see if there was a write in progress
                        If Stevedore.SavedTXString <> "" Then
                            'call the write routine
                            SendData = True
                        End If
                    End SyncLock
                    If Tock > LastTimeCheckedWrite + TimeBetweenWriteChecks
Then
                        'time to pulse the data x-mit routine
                        SendData = True
                    End If

                    If SendData = True Then
                        'write data
                        Stevedore.MainPortWriter()
                        'update time
                        LastTimeCheckedWrite = Tock
                    End If
                End If

                'UI, Speed calculation update:  synchronize!
                If Tock > LastTimeUpdated + TimeBetweenUpdates Then
                    'update the last update time
                    LastTimeUpdated = Tock

                    'Determine the Speed Pin Positive Fraction Deviation
                    'Calculates how far off the actual value is
                    PinDeviation = DesiredSpeedPinPositiveFraction -
MeasuredOutputPinPositiveFraction
                    If Math.Abs(PinDeviation) > 0.05 Then
                        'A significant Deviation existed in the last frame; log
it
                        Lumberjack.SendToLog("NOTE:  The previous Frame had a
high Speed Control Deviation:  " & PinDeviation.ToString)
                    End If

                    'calculate the speed
                    CalculatedSpeed = SpeedLambda * (SpeedPulsesUp +
SpeedPulsesDown)
                    'set speed to zero if below cutoff
                    If CalculatedSpeed < SpeedZeroCutoff Then
                        CalculatedSpeed = 0
                    End If
                    If CalculatedSpeed > ((Freq * RecentLoops) / (RecentTocks *
2.5)) Then
                        'speed is at or approaching immeasurably high values
                        Lumberjack.SendToLog("WARNING:  CURRENT SPEED IS AT OR
NEAR IMMEASURABLY HIGH VALUES; CURRENT SPEED = " & CalculatedSpeed.ToString &
"; CURRENT LIMIT = " & (0.5 * (Freq * RecentLoops) / RecentTocks).ToString)
                    End If

                    'update the speed and time arrays
                    'the values get shifted to lower indices such that the most
recent is in the highest index
                    For Regression_Counter = 0 To Regression_nmax - 1
                        'time
```

```
                        Regression_x_Tocks(Regression_Counter) =
Regression_x_Tocks(Regression_Counter + 1)
                        'speed
                        Regression_y(Regression_Counter) =
Regression_y(Regression_Counter + 1)
                    Next
                    Regression_x_Tocks(Regression_nmax) = Tock
                    Regression_y(Regression_nmax) = CalculatedSpeed
                    'the Regression_x array is time relative to now, in seconds
(it should be decreasingly negative; the highest index should have a value of
zero)
                    For Regression_Counter = 0 To Regression_nmax
                        Regression_x(Regression_Counter) =
(Regression_x_Tocks(Regression_Counter) - Tock) / Freq
                    Next

                    'determine whether or not the speed is in range
                    If (TargetSpeed > 5) And (LocalRunButtonClicked = True)
Then
                        'target speed is in the control range, see if it's
within error bounds of the target
                        If Math.Abs((CalculatedSpeed - TargetSpeed) /
TargetSpeed) < SpeedError Then
                            'speed is inside target range, count
                            'the counter stores the tock when the speed entered
the range, so
                            If (Tock - TargetSpeedAcquiredCounterTocks) >
TargetSpeedAcquiredDelayTime Then
                                'it endured the countdown and is fully within
range
                                'log it, for entry into the range
                                If TargetSpeedAcquired = False Then
                                    Lumberjack.SendToLog("Speed has entered the
Target Range; currently " & CalculatedSpeed.ToString)
                                End If
                                TargetSpeedAcquired = True
                            End If
                        Else
                            'speed is outside the target range
                            If TargetSpeedAcquired = True Then
                                'log the departure from the target range
                                Lumberjack.SendToLog("Speed has left the Target
Range; currently " & CalculatedSpeed.ToString)
                            End If
                            TargetSpeedAcquired = False
                            TargetSpeedAcquiredCounterTocks = Tock
                        End If
                    Else
                        'target speed is outside the control range or system is
not running
                        If TargetSpeedAcquired = True Then
                            'log the departure from the target range
                            Lumberjack.SendToLog("Speed has left the Target
Range; currently " & CalculatedSpeed.ToString)
                        End If
                        TargetSpeedAcquired = False
                        TargetSpeedAcquiredCounterTocks = Tock
                    End If
```

```
                    'limit the influence of the past frame's target speed pin
positive fractions
                    If (TotalTimeTocks > 0) And (TotalTimeTocks > 0.00025 *
TimeBetweenUpdates) Then
                        'can limit, set it to a fraction of the time of the
last frame in tocks
                        'do it for both the total and positive time
                        PositiveTimeTocks = PositiveTimeTocks * (0.00025 *
TimeBetweenUpdates / TotalTimeTocks)
                        'don't rewrite the following equation in case rounding
errors are significant
                        TotalTimeTocks = TotalTimeTocks * (0.00025 *
TimeBetweenUpdates / TotalTimeTocks)
                    End If

                    'Send Values
                    SyncLock Windowsill.RunBoxLock
                        'speed
                        Windowsill.DetectedSpeed = CalculatedSpeed
                        'speed in range indicator
                        Windowsill.ProperSpeed = TargetSpeedAcquired
                        'cavitation
                        Windowsill.Cavitation = CavitationDetected
                        'laser fired indication
                        If FiredLaser = True Then
                            'laser was fired
                            Windowsill.LaserFireIndicator = True
                            'clear the flag once transmitted
                            FiredLaser = False
                        End If
                        'Speed restart stuff
                        Windowsill.RestartStatus = SpeedRestartStatus
                        Windowsill.RestartTimeLeft = CLng(SpeedRestartTimeLeft)
                        'autopulsing
                        If StopAutopulsing = True Then
                            'no autopulsing
                            Windowsill.DisableAutomaticPulsing = True
                            StopAutopulsing = False
                        End If
                        'laser stage
                        Windowsill.LaserStage = LaserFireStage
                    End SyncLock

                    'Load Values
                    SyncLock Windowsill.RunBoxLock
                        'automatic pulsing
                        If (Windowsill.AutomaticPulsing = False) Or
(Windowsill.DisableAutomaticPulsing = True) Then
                            'no autopulsing the laser
                            AutoPulsing = False
                        ElseIf StopAutopulsing = True Then
                            AutoPulsing = False
                        Else
                            'enable autopulsing
                            AutoPulsing = True
                        End If
                        'check for a manually-issued laser pulse
                        LocalFireManualPulse = Windowsill.FireManualPulse
                        'control speed
```

```
                        TargetSpeed = Windowsill.DesiredSpeed
                        'pulse power
                        NextLaserPower = Windowsill.NextLaserPulsePower
                        'time between pulses
                        TimeBetweenPulses = Windowsill.DelayBetweenPulses
                        'get the pulse duration
                        LocalPulseDuration =
Windowsill.EstimatedLaserPulseDuration
                        'run status
                        LocalRunButtonClicked = Windowsill.RunButtonClicked
                        'get the restart delay
                        LocalRestartDelayTime = Windowsill.RestartDelayTime
                        'get the action when cavitation is detected
                        LocalCavitationAction = Windowsill.CavitationAction
                    End SyncLock

                    'Set the target speed to zero if there is a pause state
                    FormerRestartStatus = SpeedRestartStatus
                    'make sure that any activity gets cancelled if the status
is not correct
                    If (LocalRunButtonClicked = False) Or
(LocalCavitationAction <> 2) Then
                        SpeedRestartStatus = 0
                    End If
                    If (CavitationDetected = True) And (LocalCavitationAction =
2) Then
                        'set the speed to zero and the restart status to 1
                        'disable autopulsing
                        'and wait for the cavitation condition to end
                        TargetSpeed = 0
                        SpeedRestartStatus = 1
                        StopAutopulsing = True
                        AutoPulsing = False
                        'see if it should be logged
                        If FormerRestartStatus <> SpeedRestartStatus Then
                            'status changed, log it
                            Lumberjack.SendToLog("Speed Halting on Cavitation,
will restart")
                        End If
                    ElseIf SpeedRestartStatus = 2 Then
                        'COUNTDOWN!
                        SpeedRestartTimeLeft = SpeedRestartTimeLeft -
(CDbl(Tock - RestartDelayTimerTocks) / CDbl(Freq))
                        RestartDelayTimerTocks = Tock
                        'set target speed
                        TargetSpeed = 0
                        'see if the Countdown has completed
                        If SpeedRestartTimeLeft <= 0 Then
                            'it finished
                            SpeedRestartTimeLeft = 0
                            'change the status
                            SpeedRestartStatus = 0
                            'log the end
                            Lumberjack.SendToLog("Speed Restart Sequence
Completed, Resuming")
                        End If
                    ElseIf ((LocalCavitationAction = 2) And (SpeedRestartStatus
= 1)) And ((CavitationDetected = False) And (CalculatedSpeed <=
StopSpeedValue)) Then
```

```
                        'we're in stop and restart mode, cavitation was
detected and cleared, and the spinner has stopped
                        'move on to Phase 2
                        SpeedRestartStatus = 2
                        'start the countdown
                        SpeedRestartTimeLeft = LocalRestartDelayTime
                        RestartDelayTimerTocks = Tock
                        'log it
                        Lumberjack.SendToLog("Speed Halted and Cavitation
Cleared, will wait " & SpeedRestartTimeLeft.ToString & " s before Resuming")
                        'set target speed
                        TargetSpeed = 0
                    ElseIf ((LocalCavitationAction = 2) And (SpeedRestartStatus
= 1)) Then
                        'in a stop+restart sequence
                        'wait for it to stop
                        'maintain target speed at zero
                        TargetSpeed = 0
                    Else
                        'set the restart status to zero
                        If SpeedRestartStatus <> 0 Then
                            SpeedRestartStatus = 0
                            'log the return to normalcy
                            Lumberjack.SendToLog("Speed Restart Sequence
Ended")
                        End If
                    End If

                    'Determine the new time fraction for positive output on the
speed control pin
                    If (TargetSpeed = 0) Or (LocalRunButtonClicked = False)
Then
                        'either the control is disabled or the speed is zero
                        'set target fraction to zero
                        DesiredSpeedPinPositiveFraction = 0
                    ElseIf TargetSpeed < 0 Then
                        'a negative target speed means full speed ahead if
equal to or less than -1
                        'otherwise, it sets the pin positive fraction as its
negative
                        'i.e., if TargetSpeed = -0.25, then
DesiredSpeedPinPositiveFractio = 0.25
                        If TargetSpeed < -1 Then
                            'set target fraction to one
                            DesiredSpeedPinPositiveFraction = 1
                        Else
                            'set target fraction to the negative of the
targetspeed
                            DesiredSpeedPinPositiveFraction = -TargetSpeed
                        End If
                    Else
                        'calculate the new value between zero and one,
inclusive
                        'use an appriopriate difference in the fraction; there
are limits
                        '
                        'What this section attempts to do is
                        '   (1) Predict what the speed will be in the very near
future using a linear regression scheme
```

```
                        '   (2) Use the predicted speed in comparison with the
desired speed to find a speed difference
                        '   (3) Use the speed difference to come up with a
delta from the stored output fraction
                        '       The stored output fraction is in the Capacitor
variable, and is a sort of running average
                        '       of the output fraction on the speed pin,
simulating the phase capacitor in the electronics
                        '   (4) Within limits, determine the new output
fraction by summing the Capacitor variable with the
                        '       delta and a fraction of the Deviation from the
previous Frame's desired value

                        'Perform the Regression Calculations
                        Regression_xbar = 0
                        Regression_ybar = 0
                        Regression_s_xy = 0
                        Regression_s_xx = 0
                        Regression_s_yy = 0
                        'set the averages
                        For Regression_Counter = 0 To Regression_nmax
                            Regression_xbar = Regression_xbar + (1 /
CDbl(Regression_nmax + 1)) * Regression_x(Regression_Counter)
                            Regression_ybar = Regression_ybar + (1 /
CDbl(Regression_nmax + 1)) * Regression_y(Regression_Counter)
                        Next
                        'set the s-values
                        For Regression_Counter = 0 To Regression_nmax
                            Regression_s_xy = Regression_s_xy + (1 /
CDbl(Regression_nmax)) * (Regression_x(Regression_Counter) - Regression_xbar) *
(Regression_y(Regression_Counter) - Regression_ybar)
                            Regression_s_xx = Regression_s_xx + (1 /
CDbl(Regression_nmax)) * (Regression_x(Regression_Counter) - Regression_xbar) ^
2
                            'Regression_s_yy = Regression_s_yy + (1 /
CDbl(Regression_nmax)) * (Regression_y(Regression_Counter) - Regression_ybar) ^
2
                        Next
                        'calculate the beta-values
                        'If Regression_s_xy <> 0 Then
                        '    'no divide-by-zero
                        '    Regression_Beta1 = (Regression_s_yy -
(Regression_Delta * Regression_s_xx) + Math.Sqrt((Regression_s_yy -
(Regression_Delta * Regression_s_xx)) ^ 2 + (4 * Regression_Delta *
(Regression_s_xy ^ 2)))) / (2 * Regression_s_xy)
                        'Else
                        '    Regression_Beta1 = 0
                        '    'a divide-by-zero was prevented, and is probably
the result of a constant speed=0
                        'End If

                        'DEBUGGING ALTERATION
                        'SINCE THE ORIGINAL DEMING REGRESSION WAS PRODUCING
OCCASIONAL WEIRD RESULTS
                        'FALL BACK TO SIMPLE LINEAR REGRESSION
                        If Regression_s_xx <> 0 Then
                            'it's simple
                            'no divide by zero here
```

```
                            Regression_Beta1 = Regression_s_xy /
Regression_s_xx
                        Else
                            'this might happen once
                            Regression_Beta1 = 0
                        End If


                        Regression_Beta0 = Regression_ybar - (Regression_Beta1
* Regression_xbar)

                        'determine the predicted speed
                        'this is in the form of y = b + m*x
                        'where x is the time-value; time=0 is indexed to 'Tock'
in the Regression Calculations above
                        PredictedSpeed = Regression_Beta0 + (Regression_Beta1 *
Look_Ahead_Time)
                        'determine the difference in predicted speed and the
target speed
                        DeltaSpeed = TargetSpeed - PredictedSpeed
                        'calculate the resulting speed pin positive fraction
change
                        SpeedDeltaFraction = (SpeedDeltaCoefficient *
DeltaSpeed) + (SpeedDeltaCoefficient * (DeltaSpeed / CrossoverSpeed) ^ 3)
                        'limit the change
                        If SpeedDeltaFraction > SpeedDeltaLimiter *
TimeBetweenUpdates / Freq Then
                            'fraction change too high, cap it
                            SpeedDeltaFraction = SpeedDeltaLimiter *
TimeBetweenUpdates / Freq
                        ElseIf SpeedDeltaFraction < (-SpeedDeltaLimiter) *
TimeBetweenUpdates / Freq Then
                            'fraction change too negative, cap it
                            SpeedDeltaFraction = (-SpeedDeltaLimiter) *
TimeBetweenUpdates / Freq
                        End If

                        'Check for a Jump condition
                        'a Jump is when the speed makes a sudden, rapid, and
unexpected change
                        'it can happen with a constant output fraction
                        If Math.Abs(Regression_Beta1) >= JumpSlope Then
                            'the slope condition is met, check for the others
                            Lumberjack.SendToLog("DEBUG MSG:  Jump Condition #1
(Slope) met")
                            If ((-DeltaSpeed / TargetSpeed > SpeedError) And
(Regression_Beta1 > 0)) Or ((-DeltaSpeed / TargetSpeed < -SpeedError) And
(Regression_Beta1 < 0)) Then
                                'the Difference condition is met
                                Lumberjack.SendToLog("DEBUG MSG:  Jump
Condition #2 (Difference) met")
                                If Math.Abs((CalculatedSpeed - TargetSpeed) /
TargetSpeed) <= 3.0R * SpeedError Then
                                    'The Speed Condition is met
                                    Lumberjack.SendToLog("DEBUG MSG:  Jump
Condition #3 (Speed) met")
                                    If Tock - LastJumpTock >= TocksBetweenJumps
Then
```

```
                                        'The Delay (time between events)
condition is met
                                        Lumberjack.SendToLog("DEBUG MSG:  Jump
Condition #4 (Time) met")
                                        'The Jump conditions have been met,
apply the Jump Countermeasures
                                        'record the Jump Countermeasures event
                                        JumpCount += 1
                                        LastJumpTock = Tock
                                        Lumberjack.SendToLog("Speed Jump
Condition detected, applying Countermeasures # " & JumpCount.ToString)
                                        'apply the Countermeasures
                                        SpeedDeltaFraction = SpeedDeltaFraction
- (JumpValue * Regression_Beta1 / JumpSlope)
                                    End If
                                End If
                            End If
                        End If

                        'get the new value if the speed is not overridden
                        If SpeedOverride = False Then
                            DesiredSpeedPinPositiveFraction = Capacitor +
SpeedDeltaFraction + (0.5 * PinDeviation)
                        End If
                        'make sure the new value is within range
                        If DesiredSpeedPinPositiveFraction > 1 Then
                            'it's too high, cap it at 1
                            DesiredSpeedPinPositiveFraction = 1
                        ElseIf DesiredSpeedPinPositiveFraction < 0 Then
                            'too low, limit it to zero
                            DesiredSpeedPinPositiveFraction = 0
                        End If

                        'DEBUG OUTPUT BLOCK
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
CURRENT SPEED = " & CalculatedSpeed.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
PREDICT SPEED = " & PredictedSpeed.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:  NEW
SPD PIN F = " & DesiredSpeedPinPositiveFraction.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
DEMING BETA 0 = " & Regression_Beta0.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
DEMING BETA 1 = " & Regression_Beta1.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
DEMING S XX 1 = " & Regression_s_xx.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
DEMING S YY 1 = " & Regression_s_yy.ToString)
                        'Lumberjack.SendToLog("DEBUGGING OUTPUT MESSAGE:
DEMING S XY 1 = " & Regression_s_xy.ToString)

                    End If

                End If

                'Release Control
                Application.DoEvents()
                SyncLock CentralClass.ProgramExitLock
                    ContinueExecution = Not CentralClass.ExitProgram
```

```
            End SyncLock

        Loop


        'get the loop end time
        QueryPerformanceCounter(LoopEndTocks)

        'Shutdown Routine
        'perform preliminary shutdown tasks here

        'press stop
        SyncLock Windowsill.RunBoxLock
            Windowsill.StopAction = True
        End SyncLock

        'make sure the laser dataline is unblocked
        SyncLock Windowsill.RunBoxLock
            Windowsill.LaserTXDataCritical = False
        End SyncLock

        'set speed control and laser fire pins
        If LocalPortOpen = True Then
            'set the laser pulse and speed pins low
            'this SHOULD stop the spinner and end any laser pulsing
            SyncLock Stevedore.SpeedPortLock
                Stevedore.COMPort1.DtrEnable = False
                Stevedore.COMPort1.RtsEnable = False
            End SyncLock
        End If

        'Send Values to the Runbox
        SyncLock Windowsill.RunBoxLock
            'speed
            Windowsill.DetectedSpeed = CalculatedSpeed
            'speed in range indicator
            Windowsill.ProperSpeed = False
            'cavitation
            Windowsill.Cavitation = False
            'Speed restart stuff
            Windowsill.RestartStatus = 0
            Windowsill.RestartTimeLeft = 0
            'autopulsing
            Windowsill.DisableAutomaticPulsing = True
        End SyncLock

        If UsedAdvancedTiming = True Then
            Lumberjack.SendToLog("Disabling Advanced Timing Feature")
            'because I'm being a lazy programmer and don't want to change
the following code
            'since accuracy no longer matters at this point in execution
            'log the live fraction, if applicable
            Lumberjack.SendToLog("Current Live Fraction:  " &
(CDbl(LiveTocks - LoggedLiveTocks) / CDbl(Tock - LoggedTotalTocks)).ToString)
            'update the logged times
            LoggedLiveTocks = LiveTocks
            LoggedTotalTocks = Tock
        End If
```

```
            'wait for the correct shutdown state to complete
            Do
                'continue monitoring and reporting the speed
                SyncLock CentralClass.ProgramExitLock
                    LocalExecutionStage = CentralClass.ExecutionStage
                End SyncLock

                'Clock
                LastTime = Tock
                QueryPerformanceCounter(Tock)
                DeltaTocks = Tock - LastTime

                SpeedDetectorPinPositivePrevious = SpeedDetectorPinPositive
                CavitationPinPositivePrevious = CavitationPinPositive
                SyncLock Stevedore.SpeedPortLock
                    If Stevedore.COMPort1.IsOpen = True Then
                        'get the speed pin status
                        SpeedDetectorPinPositive =
Stevedore.COMPort1.DsrHolding
                        'get the cavitation pin status
                        'positive means the instrument can receive greater
amounts of light than the negative status
                        'which should happen when a vapor column appears
                        CavitationPinPositive = Stevedore.COMPort1.CDHolding
                        'get the speed controller pin status
                        SpeedControlPinPositive = Stevedore.COMPort1.DtrEnable
                        'get the laser fire pin status
                        LaserFirePinPositive = Stevedore.COMPort1.RtsEnable
                    Else
                        'the port is not open, close the program
                        ContinueExecution = False
                    End If
                End SyncLock
                If ContinueExecution = False Then
                    'signal a program exit
                    SyncLock CentralClass.ProgramExitLock
                        CentralClass.ExitProgram = True
                    End SyncLock
                End If

                'process the input pin status meanings
                'if the speed pin's current state and previous state are
different, increment the appropriate counter
                If SpeedDetectorPinPositive <> SpeedDetectorPinPositivePrevious
Then
                    'speed pulse received, see if pin went high
                    If SpeedDetectorPinPositive = True Then
                        'the pin went positive
                        'increment the speedpulsesup counter
                        SpeedPulsesUp += 1
                    Else
                        'the pin went negative
                        'increment the other counter
                        SpeedPulsesDown += 1
                    End If
                End If
                'do some speed processing:  calculate the remaining 'sum' of
pulses
                RemainingFraction = Math.Exp(DecayConstant * CDbl(DeltaTocks))
```

```
                SpeedPulsesUp = SpeedPulsesUp * RemainingFraction
                SpeedPulsesDown = SpeedPulsesDown * RemainingFraction

                'process the cavitation status
                CavitationDetectedPrevious = CavitationDetected
                If CavitationPinPositive = True Then
                    'cavitation detected?  give it a timeout to make sure
                    If CavitationPinPositive <> CavitationPinPositivePrevious
Then
                        'new signal, start the countdown
                        CavitationStartTime = Tock
                    Else
                        'continuing signal, see if the countdown has completed
                        If Tock > (CavitationStartTime +
CavitationTimeoutTocks) Then
                            'countdown has finished without sensing a non-
cavitation condition in the meantime
                            CavitationDetected = True
                            'log it if it's a first-time event
                            If CavitationDetectedPrevious = False Then
                                'first-time event, log the detection
                                Lumberjack.SendToLog("Cavitation Detected!
Speed is " & CalculatedSpeed.ToString)
                            End If
                        End If
                    End If
                Else
                    'Cavitation not detected
                    CavitationDetected = False
                    'log termination of cavitation condition if it existed
                    If CavitationDetectedPrevious = True Then
                        'cavitation condition just stopped, log it
                        Lumberjack.SendToLog("Cavitation Condition No Longer
Detected")
                    End If
                End If

                'Read the Serial Port data, if it's time
                If Tock > LastTimeCheckedRead + TimeBetweenReads Then
                    'check for data in the serial port
                    Stevedore.MainPortReader()
                    'update checked time
                    LastTimeCheckedRead = Tock
                End If

                'Transmit data through the Speed Port (Mode 1 only)
                If LocalProgramMode = 1 Then
                    SendData = False
                    'check for write
                    SyncLock Stevedore.MainPortWriterLock
                        'see if there was a write in progress
                        If Stevedore.SavedTXString <> "" Then
                            'call the write routine
                            SendData = True
                        End If
                    End SyncLock
                    If Tock > LastTimeCheckedWrite + TimeBetweenWriteChecks
Then
                        'time to pulse the x-mit routine
```

```
                    SendData = True
                End If

                If SendData = True Then
                    'write data
                    Stevedore.MainPortWriter()
                    'update time
                    LastTimeCheckedWrite = Tock
                End If
            End If

            'calculate the speed
            CalculatedSpeed = SpeedLambda * (SpeedPulsesUp +
SpeedPulsesDown)
            'set speed to zero if below cutoff
            If CalculatedSpeed < SpeedZeroCutoff Then
                CalculatedSpeed = 0
            End If

            'update logged speed if the conditions are met
            'those include time and speed change
            If (Tock - LastRecordedSpeedTime) / Freq >
(MaxTimeBetweenSpeedRecordingTicks / 10000000) Then
                'it's time to log the speed
                Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                'update the speed recording time and value
                LastRecordedSpeed = CalculatedSpeed
                LastRecordedSpeedTime = Tock
            ElseIf CalculatedSpeed > StopSpeedValue Then
                'it is not stopped
                'check for standard delta
                If Math.Abs(CalculatedSpeed - LastRecordedSpeed) >
MaxDeltaSpeedRecording Then
                    'Speed is different enough to update in the log
                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                    'update the speed recording time and value
                    LastRecordedSpeed = CalculatedSpeed
                    LastRecordedSpeedTime = Tock
                End If
            Else
                'it's currently stopped, see if it was before
                If LastRecordedSpeed > StopSpeedValue Then
                    'wasn't stopped last time, log it
                    Lumberjack.SendToLog("Speed Effectively Stopped")
                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                    'update the speed recording time and value
                    LastRecordedSpeed = CalculatedSpeed
                    LastRecordedSpeedTime = Tock
                End If
            End If

            'Update display
            If Tock > LastTimeUpdated + TimeBetweenUpdates Then
                'update the last update time
                LastTimeUpdated = Tock
                'Send Values
```

```
                    SyncLock Windowsill.RunBoxLock
                        'speed
                        Windowsill.DetectedSpeed = CalculatedSpeed
                        'speed in range indicator
                        Windowsill.ProperSpeed = TargetSpeedAcquired
                        'cavitation
                        Windowsill.Cavitation = CavitationDetected
                        'laser fired indication
                        If FiredLaser = True Then
                            'laser was fired
                            Windowsill.LaserFireIndicator = True
                            'clear the flag once transmitted
                            FiredLaser = False
                        End If
                        'Speed restart stuff
                        Windowsill.RestartStatus = SpeedRestartStatus
                        Windowsill.RestartTimeLeft = CLng(SpeedRestartTimeLeft)
                        'autopulsing
                        If StopAutopulsing = True Then
                            'no autopulsing
                            Windowsill.DisableAutomaticPulsing = True
                            StopAutopulsing = False
                        End If
                    End SyncLock
                End If

                'Release Control
                Application.DoEvents()

            Loop While LocalExecutionStage < 1010


            'wait for spinner to stop, or give an error message
            SpinnerStopTimeoutCounter = Tock
            Do
                'Clock
                LastTime = Tock
                QueryPerformanceCounter(Tock)
                DeltaTocks = Tock - LastTime

                SpeedDetectorPinPositivePrevious = SpeedDetectorPinPositive
                CavitationPinPositivePrevious = CavitationPinPositive
                SyncLock Stevedore.SpeedPortLock
                    If Stevedore.COMPort1.IsOpen = True Then
                        'get the speed pin status
                        SpeedDetectorPinPositive =
Stevedore.COMPort1.DsrHolding
                        'get the cavitation pin status
                        'positive means the instrument can receive greater
amounts of light than the negative status
                        'which should happen when a vapor column appears
                        CavitationPinPositive = Stevedore.COMPort1.CDHolding
                        'get the speed controller pin status
                        SpeedControlPinPositive = Stevedore.COMPort1.DtrEnable
                        'get the laser fire pin status
                        LaserFirePinPositive = Stevedore.COMPort1.RtsEnable
                    Else
                        'the port is not open, close the program
                        ContinueExecution = False
```

```
                    End If
                End SyncLock
                If ContinueExecution = False Then
                    'signal a program exit
                    SyncLock CentralClass.ProgramExitLock
                        CentralClass.ExitProgram = True
                    End SyncLock
                End If

                'process the input pin status meanings
                'if the speed pin's current state and previous state are
different, increment the appropriate counter
                If SpeedDetectorPinPositive <> SpeedDetectorPinPositivePrevious
Then
                    'speed pulse received, see if pin went high
                    If SpeedDetectorPinPositive = True Then
                        'the pin went positive
                        'increment the speedpulsesup counter
                        SpeedPulsesUp += 1
                    Else
                        'the pin went negative
                        'increment the other counter
                        SpeedPulsesDown += 1
                    End If
                End If
                'do some speed processing:  calculate the remaining 'sum' of
pulses
                RemainingFraction = Math.Exp(DecayConstant * CDbl(DeltaTocks))
                SpeedPulsesUp = SpeedPulsesUp * RemainingFraction
                SpeedPulsesDown = SpeedPulsesDown * RemainingFraction

                'process the cavitation status
                CavitationDetectedPrevious = CavitationDetected
                If CavitationPinPositive = True Then
                    'cavitation detected?  give it a timeout to make sure
                    If CavitationPinPositive <> CavitationPinPositivePrevious
Then
                        'new signal, start the countdown
                        CavitationStartTime = Tock
                    Else
                        'continuing signal, see if the countdown has completed
                        If Tock > (CavitationStartTime +
CavitationTimeoutTocks) Then
                            'countdown has finished without sensing a non-
cavitation condition in the meantime
                            CavitationDetected = True
                            'log it if it's a first-time event
                            If CavitationDetectedPrevious = False Then
                                'first-time event, log the detection
                                Lumberjack.SendToLog("Cavitation Detected!
Speed is " & CalculatedSpeed.ToString)
                            End If
                        End If
                    End If
                Else
                    'Cavitation not detected
                    CavitationDetected = False
                    'log termination of cavitation condition if it existed
                    If CavitationDetectedPrevious = True Then
```

```
                        'cavitation condition just stopped, log it
                        Lumberjack.SendToLog("Cavitation Condition No Longer
Detected")
                    End If
                End If

                'Read the Serial Port data, if it's time
                If Tock > LastTimeCheckedRead + TimeBetweenReads Then
                    'check for data in the serial port
                    Stevedore.MainPortReader()
                    'update checked time
                    LastTimeCheckedRead = Tock
                End If

                'Transmit data through the Speed Port (Mode 1 only)
                If LocalProgramMode = 1 Then
                    SendData = False
                    'check for write
                    SyncLock Stevedore.MainPortWriterLock
                        'see if there was a write in progress
                        If Stevedore.SavedTXString <> "" Then
                            'call the write routine
                            SendData = True
                        End If
                    End SyncLock
                    If Tock > LastTimeCheckedWrite + TimeBetweenWriteChecks
Then
                        'time to pulse the x-mit routine
                        SendData = True
                    End If

                    If SendData = True Then
                        'write data
                        Stevedore.MainPortWriter()
                        'update time
                        LastTimeCheckedWrite = Tock
                    End If
                End If

                'calculate the speed
                CalculatedSpeed = SpeedLambda * (SpeedPulsesUp +
SpeedPulsesDown)
                'set speed to zero if below cutoff
                If CalculatedSpeed < SpeedZeroCutoff Then
                    CalculatedSpeed = 0
                End If

                'update logged speed if the conditions are met
                'those include time and speed change
                If (Tock - LastRecordedSpeedTime) / Freq >
(MaxTimeBetweenSpeedRecordingTicks / 10000000) Then
                    'it's time to log the speed
                    Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                    'update the speed recording time and value
                    LastRecordedSpeed = CalculatedSpeed
                    LastRecordedSpeedTime = Tock
                ElseIf CalculatedSpeed > StopSpeedValue Then
                    'it is not stopped
```

```
                    'check for standard delta
                    If Math.Abs(CalculatedSpeed - LastRecordedSpeed) >
MaxDeltaSpeedRecording Then
                        'Speed is different enough to update in the log
                        Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                        'update the speed recording time and value
                        LastRecordedSpeed = CalculatedSpeed
                        LastRecordedSpeedTime = Tock
                    End If
                Else
                    'it's currently stopped, see if it was before
                    If LastRecordedSpeed > StopSpeedValue Then
                        'wasn't stopped last time, log it
                        Lumberjack.SendToLog("Speed Effectively Stopped")
                        Lumberjack.SendToLog("Current Speed:  " &
CalculatedSpeed.ToString)
                        'update the speed recording time and value
                        LastRecordedSpeed = CalculatedSpeed
                        LastRecordedSpeedTime = Tock
                    End If
                End If

                'Update display
                If Tock > LastTimeUpdated + TimeBetweenUpdates Then
                    'update the last update time
                    LastTimeUpdated = Tock
                    'Send Values
                    SyncLock Windowsill.RunBoxLock
                        'speed
                        Windowsill.DetectedSpeed = CalculatedSpeed
                        'speed in range indicator
                        Windowsill.ProperSpeed = TargetSpeedAcquired
                        'cavitation
                        Windowsill.Cavitation = CavitationDetected
                        'laser fired indication
                        If FiredLaser = True Then
                            'laser was fired
                            Windowsill.LaserFireIndicator = True
                            'clear the flag once transmitted
                            FiredLaser = False
                        End If
                        'Speed restart stuff
                        Windowsill.RestartStatus = SpeedRestartStatus
                        Windowsill.RestartTimeLeft = CLng(SpeedRestartTimeLeft)
                        'autopulsing
                        If StopAutopulsing = True Then
                            'no autopulsing
                            Windowsill.DisableAutomaticPulsing = True
                            StopAutopulsing = False
                        End If
                    End SyncLock
                End If

                'Release Control
                Application.DoEvents()

                If (Tock - SpinnerStopTimeoutCounter) / Freq > 30 Then
                    'timeout waiting for the spinner to halt
```

```
                    'log it and move on
                    Lumberjack.SendToLog("Timeout waiting for spinner to halt;
speed is " & CalculatedSpeed.ToString)
                    Exit Do
                End If
            Loop While CalculatedSpeed > StopSpeedValue

            Application.DoEvents()

            'Close the Port
            Try
                'if the port was opened, close it
                If LocalPortOpen = True Then
                    'log it
                    Lumberjack.SendToLog("Attempting to Close the " &
PortUseString & " (" & LocalPortName & ")")
                    'send final string to the port, if there is one
                    'get any data
                    If LocalProgramMode = 1 Then
                        'this thread controls the port, get final data to send
                        SyncLock Stevedore.MainPortWriterLock
                            'get leftover string, if any
                            LocalTXString = Stevedore.SavedTXString
                            Stevedore.SavedTXString = ""
                        End SyncLock

                        SyncLock Windowsill.RunBoxLock
                            'get any last commands
                            LocalTXString = LocalTXString &
Windowsill.CommandToLaser
                            Windowsill.CommandToLaser = ""
                        End SyncLock

                        'append a newline character
                        LocalTXString = LocalTXString & LaserEOL
                    Else
                        'laser either on another port, disconnected, or ignored
                        'send a newline character
                        LocalTXString = LaserEOL
                    End If

                    Try
                        'transmit any data
                        If LocalTXString <> "" Then
                            SyncLock Stevedore.SpeedPortLock
                                Stevedore.COMPort1.Write(LocalTXString)
                            End SyncLock
                        End If
                    Catch FinishedWriteException As Exception
                        'log the issue
                        Lumberjack.SendToLog("Exception in final write on " &
LocalPortName & ":  " & FinishedWriteException.Message)
                    End Try

                    Thread.Sleep(250)
                    'get all the data from the port, then close it
                    SyncLock Stevedore.MainPortReaderLock
                        'collect any text waiting to be logged
                        LocalRXString = Stevedore.TempRXString
```

```
                    Stevedore.TempRXString = ""
                End SyncLock

                SyncLock Stevedore.SpeedPortLock
                    If Stevedore.COMPort1.IsOpen = True Then
                        'get the last bit of data from the port
                        LocalRXString = LocalRXString &
Stevedore.COMPort1.ReadExisting()
                        'the port is open, close it
                        Stevedore.COMPort1.Close()
                        Stevedore.COMPort1.Dispose()
                    End If
                    LocalPortOpen = Stevedore.COMPort1.IsOpen
                End SyncLock
                Thread.Sleep(1000)
                'log success/failure to close the port
                If LocalPortOpen = True Then
                    'port is still open, log an error
                    Lumberjack.SendToLog(PortUseString & " (" &
LocalPortName & ") did not close successfully")
                Else
                    'port closed
                    Lumberjack.SendToLog("Successfully Closed the " &
PortUseString & " (" & LocalPortName & ")")
                End If
            End If

        Catch SerialPortClosureProblem As Exception
            'log it
            Lumberjack.SendToLog("Exception while attempting to close the "
& PortUseString & " (" & LocalPortName & "):  " & vbNewLine &
SerialPortClosureProblem.Message)
        End Try

        'send the output where it belongs
        If LocalRXString <> "" Then
            'log the final rx string
            Lumberjack.SendToLog("Final Text Received from the " &
PortUseString & " (" & LocalPortName & "):  " & LocalRXString)
            'mirror it on the runbox
            SyncLock Windowsill.RunBoxLock
                Windowsill.LaserRX = Windowsill.LaserRX & LocalRXString
            End SyncLock
            'clear the string
            LocalRXString = ""
        End If

        Thread.Sleep(3000)
        'log final comments
        Lumberjack.SendToLog(Freq.ToString & " Tocks/second")
        Lumberjack.SendToLog(LiveTocks.ToString & " Tocks Live")
        Lumberjack.SendToLog(AllTocks.ToString & " Tocks Total")
        If AllTocks <> 0 Then
            'prevent divide by zero errors
            Lumberjack.SendToLog((100 * LiveTocks / AllTocks).ToString & "
% Live Time")
        End If
        If TotalLoops <> 0 Then
            'prevent divide by zero
```

```
                Lumberjack.SendToLog((AllTocks / TotalLoops).ToString & "
Tocks/MainLoop")
            End If
            'log the live fraction, if applicable
            If UsedAdvancedTiming = True Then
                'advanced timing has been applied
                Lumberjack.SendToLog("Number of Applications of the Advanced
Timing Feature:  " & AdvancedTimingCounts.ToString)
            End If
            'log termination
            Lumberjack.SendToLog("Speed Port Module Stopping")

            'now perform final shutdown tasks
            Lumberjack.SendToLog("Kernel Sanders Says " & TotalLoops & "
Loops")
            'now exit the thread

            Lumberjack.SendToLog("DEBUG MSG:  LookAheadTocks = " &
LookAheadTocks.ToString)

        Catch ThreadNeededKilling As ThreadAbortException
            'the thread was aborted
            Application.ExitThread()
        Catch BigException As Exception
            'something unhandled occurred, exit the program
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End Try

        Application.ExitThread()

    End Sub



    Private Shared Sub MainPortReader()
        'performs the read operations for the speed/laser port

        Dim LocalPortOpen As Boolean = False
        Dim LastRXTimeTicks As Long = 0
        Dim CharactersRead As Integer = 0
        Dim LocalRXChar(0) As Char
        Dim PreviousCharacterNewline As Boolean = False
        Dim LocalRXString As String = ""
        Dim CharactersRemain As Boolean = False
        Dim RXTimeoutToLogTicks As Long = 2500000
        Dim RXTag As String = ""


        'get variable data
        LocalRXChar(0) = CChar("")
        SyncLock Longshoreman.LaserPortLock
```

```
            If Longshoreman.LaserControlMode = 1 Then
                'laser on the speed control port
                RXTag = "RX from Laser:  "
            Else
                'laser is on another port
                RXTag = "RX from Speed Control Port:  "
            End If
        End SyncLock

        'load saved values
        SyncLock Stevedore.MainPortReaderLock
            LastRXTimeTicks = Stevedore.LastRXTimeTicksSaved
            PreviousCharacterNewline = Stevedore.NewlineRX
            LocalRXString = Stevedore.TempRXString
        End SyncLock

        'get data from the port
        SyncLock Stevedore.SpeedPortLock
            LocalPortOpen = Stevedore.COMPort1.IsOpen
        End SyncLock

        If LocalPortOpen = True Then
            'grab the characters buffered by the serialport one at a time
            Do
                'read one character from the port, if there are any
                SyncLock Stevedore.SpeedPortLock
                    If Stevedore.COMPort1.BytesToRead > 0 Then
                        CharactersRead = Stevedore.COMPort1.Read(LocalRXChar,
0, 1)
                        'remember the last rx read time
                        LastRXTimeTicks = Now.Ticks
                    Else
                        'no characters read
                        CharactersRead = 0
                    End If
                End SyncLock
                If CharactersRead > 0 Then
                    'characters were actually read, process the data
                    'check for cr or lf
                    If (LocalRXChar(0).ToString = vbCr) Or
(LocalRXChar(0).ToString = vbLf) Then
                        'current character is a newline sort of thing
                        If PreviousCharacterNewline = True Then
                            'previous character was a newline sort of character
                            'so is the current line
                            'log it all and clear the previouscharacternewline
condition
                            PreviousCharacterNewline = False
                            LocalRXString = LocalRXString &
LocalRXChar(0).ToString
                            'send it all to the runbox
                            SyncLock Windowsill.RunBoxLock
                                Windowsill.LaserRX = Windowsill.LaserRX &
LocalRXString
                            End SyncLock
                            'log it
                            Lumberjack.SendToLog(RXTag & LocalRXString)
                            'clear the string
                            LocalRXString = ""
```

```
                        Else
                            'previous character was NOT a newline character
                            'but the current line is
                            'append it to the local rx string, but wait before
logging
                            PreviousCharacterNewline = True
                            LocalRXString = LocalRXString & LocalRXChar(0)
                        End If
                    ElseIf PreviousCharacterNewline = True Then
                        'previous but not current character was a newline
                        'send the previous data where they all belong
                        'send it all to the runbox
                        SyncLock Windowsill.RunBoxLock
                            Windowsill.LaserRX = Windowsill.LaserRX &
LocalRXString
                        End SyncLock
                        'log it
                        Lumberjack.SendToLog(RXTag & LocalRXString)
                        'clear the local rx string and put the current
character in it
                        LocalRXString = LocalRXChar(0).ToString
                        PreviousCharacterNewline = False
                    Else
                        'neither the current nor the last character were
newlines
                        'append the received character to the local rx string
                        LocalRXString = LocalRXString & LocalRXChar(0).ToString
                    End If
                    '
                End If

                'check for remaining characters
                SyncLock Stevedore.SpeedPortLock
                    If Stevedore.COMPort1.BytesToRead > 0 Then
                        'data remains
                        CharactersRemain = True
                    Else
                        CharactersRemain = False
                        'no data left
                    End If
                End SyncLock
            Loop While CharactersRemain = True
        End If

        'check for time to dump the rx data to the log
        'only if it hasn't received anything in the timeout period
        If (Now.Ticks - LastRXTimeTicks) > RXTimeoutToLogTicks Then
            'last rx time was more than one timeout ago
            'if there's data to log, log it
            If LocalRXString <> "" Then
                'stuff to log
                'send it all to the runbox
                SyncLock Windowsill.RunBoxLock
                    Windowsill.LaserRX = Windowsill.LaserRX & LocalRXString
                End SyncLock
                'log it
                Lumberjack.SendToLog(RXTag & LocalRXString)
                'clear the string
                LocalRXString = ""
```

```
            End If
        End If

        'save values
        SyncLock Stevedore.MainPortReaderLock
            Stevedore.LastRXTimeTicksSaved = LastRXTimeTicks
            Stevedore.NewlineRX = PreviousCharacterNewline
            Stevedore.TempRXString = LocalRXString
        End SyncLock

    End Sub



    Private Shared Sub MainPortWriter()
        'Performs the write operations when the speed controller port is shared
with the laser

        Dim LocalTXString As String = ""
        Dim LocalPortEncoding As System.Text.Encoding
        Dim FoundLimit As Boolean = False
        Dim SendLength As Integer = 0
        Dim ShortString As String = ""
        Dim BytesToSend() As Byte
        Dim NumberBytes As Integer = 0
        Dim SendRefObj As Object = 0
        Dim WriteNumber As Integer = 0
        Dim ClearForLaser As Boolean = False
        Dim RecentTX As Boolean = False

        'load values
        SyncLock CentralClass.FileOpsLock
            LocalPortEncoding = CentralClass.ComPortEncoding
        End SyncLock
        'load saved values
        SyncLock Stevedore.MainPortReaderLock
            LocalTXString = Stevedore.SavedTXString
            WriteNumber = Stevedore.SavedWriteNumber
        End SyncLock

        'tell fire control to hold off momentarily
        SyncLock Stevedore.SpeedPortLock
            Stevedore.ClearToFireLaser = False
        End SyncLock

        'get any data
        RecentTX = False
        SyncLock Windowsill.RunBoxLock
            LocalTXString = LocalTXString & Windowsill.CommandToLaser
            'got the data, clear the shared variable
            Windowsill.CommandToLaser = ""
        End SyncLock
        'only send when the output buffer is empty to ensure it goes in the
correct order
        If LocalTXString <> "" Then
            'set the transmission flag for later use
            RecentTX = True
            'check for an empty buffer
            SyncLock Stevedore.SpeedPortLock
```

```
                If Stevedore.COMPort1.BytesToWrite = 0 Then
                    'empty buffer, write data
                    'localportencoding
                    If LocalPortEncoding.GetByteCount(LocalTXString) >
Stevedore.COMPort1.WriteBufferSize Then
                        'too many characters
                        'split the write in half in a loop until it fits
                        FoundLimit = False
                        SendLength = LocalTXString.Length
                        Do
                            SendLength = CInt(SendLength / 2)
                            'see if the max bytes from that can be bufferred
                            If LocalPortEncoding.GetMaxByteCount(SendLength) <
Stevedore.COMPort1.WriteBufferSize Then
                                'it works
                                FoundLimit = True
                            ElseIf SendLength < 10 Then
                                'it should be longer, something is really wrong
                                'proceed anyway
                                FoundLimit = True
                            End If
                        Loop While FoundLimit = False
                        'get the string to send
                        ShortString = LocalTXString.Substring(0, SendLength)
                        'and remove it from the local tx string
                        LocalTXString = LocalTXString.Remove(0, SendLength)
                        'encode the string
                        BytesToSend = LocalPortEncoding.GetBytes(ShortString)
                        NumberBytes = BytesToSend.Length
                        'now it's ready to write
                    Else
                        'everything can fit in the buffer immediately
                        'encode the string
                        BytesToSend = LocalPortEncoding.GetBytes(LocalTXString)
                        NumberBytes = BytesToSend.Length
                        'clear the local string
                        LocalTXString = ""
                        'ready to send
                    End If
                    'write it to the serial port output asynchronously
                    SendRefObj = CObj(WriteNumber)
                    Stevedore.COMPort1.BaseStream.BeginWrite(BytesToSend, 0,
NumberBytes, AddressOf Stevedore.FinishedComPort1Send, SendRefObj)
                    WriteNumber += 1
                End If
            End SyncLock
        End If

        'save values
        SyncLock Stevedore.MainPortWriterLock
            Stevedore.SavedTXString = LocalTXString
            Stevedore.SavedWriteNumber = WriteNumber
        End SyncLock

        'Update Laser Fire Control
        ClearForLaser = True
        'check for request to fire
        If ClearForLaser = True Then
            SyncLock Windowsill.RunBoxLock
```

```
                If Windowsill.LaserTXDataCritical = False Then
                    'laser not in firing sequence, laser is not clear to fire
                    ClearForLaser = False
                End If
            End SyncLock
        End If
        'check saved data
        If ClearForLaser = True Then
            SyncLock Stevedore.MainPortWriterLock
                If Stevedore.SavedTXString <> "" Then
                    'there's saved data to be transmitted, laser is not clear
to fire
                    ClearForLaser = False
                End If
            End SyncLock
        End If
        'check available data
        If ClearForLaser = True Then
            SyncLock Windowsill.RunBoxLock
                If Windowsill.CommandToLaser <> "" Then
                    'there's data left to be loaded and transmitted, laser is
not clear to fire
                    ClearForLaser = False
                End If
            End SyncLock
        End If
        'check bufferred data
        If ClearForLaser = True Then
            SyncLock Stevedore.SpeedPortLock
                If Stevedore.COMPort1.BytesToWrite <> 0 Then
                    'there's bufferred data in transmission, laser is not clear
to fire
                    ClearForLaser = False
                End If
            End SyncLock
        End If
        If RecentTX = True Then
            'don't signal a clear line just yet
            ClearForLaser = False
        End If
        'send value to fire control
        SyncLock Stevedore.SpeedPortLock
            Stevedore.ClearToFireLaser = ClearForLaser
        End SyncLock

    End Sub



    Private Shared Sub FinishedComPort1Send(ByVal TXResult As
System.IAsyncResult)
        'gets called in asynchronous writes to the speed/laser port
        'when they finish transmitting data

        Try
            SyncLock Stevedore.SpeedPortLock
                'end the (completed) write
                Stevedore.COMPort1.BaseStream.EndWrite(TXResult)
            End SyncLock
```

```
        Catch ex As Exception
            'something went wrong
            'log the error
            Lumberjack.SendToLog("Exception in the result of transmission to
laser:  " & ex.Message)
            Lumberjack.SendToLog("Exception Data:  " &
TXResult.AsyncState.ToString())
        End Try

    End Sub


End Class
```

B.18 TESTENDER.DESIGNER.VB

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class TestEnder
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(TestEnder))
        Me.EndButton = New System.Windows.Forms.Button
        Me.Label1 = New System.Windows.Forms.Label
        Me.SuspendLayout()
        '
        'EndButton
        '
        Me.EndButton.Location = New System.Drawing.Point(12, 32)
        Me.EndButton.Name = "EndButton"
        Me.EndButton.Size = New System.Drawing.Size(109, 23)
        Me.EndButton.TabIndex = 0
        Me.EndButton.Text = "End"
        Me.EndButton.UseVisualStyleBackColor = True
        '
        'Label1
```

```
        '
        Me.Label1.AutoSize = True
        Me.Label1.Location = New System.Drawing.Point(13, 9)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(106, 13)
        Me.Label1.TabIndex = 1
        Me.Label1.Text = "Click to End Program"
        '
        'TestEnder
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(133, 67)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.EndButton)
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
        Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
        Me.MaximizeBox = False
        Me.MinimizeBox = False
        Me.Name = "TestEnder"
        Me.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide
        Me.Text = "Exiter"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Friend WithEvents EndButton As System.Windows.Forms.Button
    Friend WithEvents Label1 As System.Windows.Forms.Label
End Class
```

B.19 TESTENDER.VB

```
Public Class TestEnder
    'A debugging-oriented form that can only initiate a program shutdown



    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles EndButton.Click
        'Ends the program

        SyncLock CentralClass.ProgramExitLock
            CentralClass.ExitProgram = True
        End SyncLock
        Lumberjack.SendToLog("Program Shutdown Initiated")
        Me.Hide()

    End Sub



    Private Sub TestEnder_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        'Do not allow the form to be closed

        If e.CloseReason <> CloseReason.None Then
            e.Cancel = True
```

```
        End If

    End Sub


End Class
```

B.20 WINDOWSILL.VB

```
Option Explicit On
Option Strict On

Friend NotInheritable Class Windowsill

    'This class houses the thread, code, and related variables that operate the
RunBox
    '
    '
    Friend Shared Gooey As New Thread(AddressOf Windowsill.UserController)
    '
    '
    Private Shared FirstOrders As New Thread(AddressOf Windowsill.LaserInitTX)
    Private Shared InitComplete As Boolean = False
    '
    '
    Friend Shared LaserRX As String = ""
    Friend Shared CommandToLaser As String = ""
    Friend Shared LaserTXDataCritical As Boolean = False
    Friend Shared FireManualPulse As Boolean = False
    Friend Shared LogWritten As String = ""
    Friend Shared SpeedControlReady As Boolean = False
    Friend Shared LaserPortReady As Boolean = False
    Friend Shared RunBoxReady As Boolean = False
    Friend Shared DetectedSpeed As Double = 0
    Friend Shared ProperSpeed As Boolean = False
    Friend Shared Cavitation As Boolean = False
    Friend Shared LaserControlsUpdated As Boolean = False
    Friend Shared LaserFireIndicator As Boolean = False
    Friend Shared NextLaserPulsePower As Double = 0
    Friend Shared EstimatedLaserPulseDuration As Double = 0.000000001
    Friend Shared DelayBetweenPulses As Double = 1
    Friend Shared AutomaticPulsing As Boolean = False
    Friend Shared DisableAutomaticPulsing As Boolean = False
    Friend Shared SpeedControlsUpdated As Boolean = False
    Friend Shared DesiredSpeed As Double = 0
    Friend Shared RestartDelayTime As Double = 10
    Friend Shared CavitationAction As Long = 1
    Friend Shared RunButtonClicked As Boolean = False
    Friend Shared FinishedStartup As Boolean = False
    Friend Shared RestartStatus As Long = 0
    Friend Shared RestartTimeLeft As Long = 0
    Friend Shared StopAction As Boolean = False
    Friend Shared LaserStage As Long = 0
    Friend Shared RunBoxLock As New Object
```

```
    Private Shared Sub UserController()
        'User Interface routine
        'Handles I/O, updates on the RunBox

        Dim LocalExit As Boolean
        Dim MaxCharacters As Long
        Dim ProgramMode As Long
        Dim StartupDone As Boolean
        Dim LocalExecutionStage As Long
        Dim LastLaserPulseTime As Long
        Dim LaserPulseFlashDurationTicks As Long
        Dim LocalLaserFired As Boolean
        Dim ShowFireLaser As Boolean
        Dim LocalLaserPulsePower As Double
        Dim DetectedCavitation As Boolean
        Dim SpeedInRange As Boolean
        Dim LocalSpeed As Double
        Dim HighlightLaserReady As Boolean
        Dim LocalDisablePulsing As Boolean
        Dim LocalSetSpeed As Double
        Dim LocalRestartDelay As Double
        Dim LocalPulseDuration As Double
        Dim LocalPulseDelay As Double
        Dim LocalRestartStatus As Long
        Dim LocalRestartTimeLeft As Long
        Dim LocalRunButtonClicked As Boolean
        Dim StatusString As String
        Dim LocalCavitationBehavior As Long
        Dim LocalStopAction As Boolean
        Dim LocalLaserStage As Long = 0
        Dim LaserEOL As String = ""
        Dim LaserInitTimeout As Long = 1200000000
        Dim LaserInitTime As Long = 0

        Try
            'Program Startup
            Thread.CurrentThread.Name = "Gooey"
            StartupDone = False
            LocalLaserFired = False
            ShowFireLaser = False
            StatusString = ""
            FirePulse.Hide()
            InsertComment.Hide()
            RunBox.Hide()
            SendLaserCommand.Hide()
            'set the duration it ticks (=100 ns/tick) that the "FIRE LASER!"
signal is shown
            LaserPulseFlashDurationTicks = 2500000
            'first, set up the RunBox
            RunBox.FireLaserBox.BackColor = System.Drawing.SystemColors.Control
            'Crimson while firing
            RunBox.FireLaserBox.ForeColor =
System.Drawing.SystemColors.ControlLight
            'Yellow while firing
            'set the maximum number of characters in the log/laser output trace
            MaxCharacters = 4096
            'now set status, date, and time
            RunBox.DateBox.Text = Format(Now(), "dddd, MMMM d, yyyy")
```

```
            RunBox.TimeBox.Text = Format(Now(), "HH:mm:ss")
            RunBox.CurrentStatusBox.Text = "Starting Up"
            RunBox.CurrentSpeedBox.Text = "Not Available"
            'show output log filename
            SyncLock CentralClass.FileOpsLock
                RunBox.OutputFileBox.Text =
System.IO.Path.GetFullPath(CentralClass.Filename)
            End SyncLock
            RunBox.OutputFileBox.AutoEllipsis = True
            'Append the Log Trace Box with any current log text, then clear the
shared variable
            SyncLock Windowsill.RunBoxLock
                RunBox.LogTrace.AppendText(Windowsill.LogWritten)
                Windowsill.LogWritten = ""
                'set the initial state of the run button and startup
                Windowsill.RunButtonClicked = False
                Windowsill.FinishedStartup = False
            End SyncLock
            RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
            RunBox.LogTrace.ScrollToCaret()

            'disable controls
            RunBox.ButtonSetSpeed.Enabled = False
            RunBox.ButtonStartSpeed.Enabled = False
            RunBox.ButtonStopSpeed.Enabled = False
            RunBox.ButtonSetLaser.Enabled = False
            RunBox.ButtonFireLaser.Enabled = False
            RunBox.ButtonSendLaserCommand.Enabled = False
            RunBox.RestartDelayInputBox.Enabled = False
            RunBox.SetSpeedInputBox.Enabled = False
            RunBox.OnCavitationContinue.Enabled = False
            RunBox.OnCavitationStop.Enabled = False
            RunBox.OnCavitationStopAndRestart.Enabled = False
            RunBox.AutomaticPulsingTrue.Enabled = False
            RunBox.AutomaticPulsingFalse.Enabled = False
            RunBox.LaserRxBox.Enabled = False
            RunBox.EstimatedDurationInputBox.Enabled = False
            RunBox.LaserDelayInputBox.Enabled = False

            'get the laser eol sequence
            SyncLock CentralClass.FileOpsLock
                LaserEOL = CentralClass.LaserNewlineString
            End SyncLock

            RunBox.Show()
            LocalExit = False
            Application.DoEvents()

            'get the operating mode
            SyncLock Longshoreman.LaserPortLock
                ProgramMode = Longshoreman.LaserControlMode
            End SyncLock

            'announce the readiness of the RunBox
            SyncLock Windowsill.RunBoxLock
                Windowsill.RunBoxReady = True
            End SyncLock

            'wait for other startup tasks to complete
```

```
            Do
                'check for speed control readiness
                SyncLock Windowsill.RunBoxLock
                    If (Windowsill.LaserPortReady = True) And
(Windowsill.SpeedControlReady) = True Then
                        'other threads ready, proceed to main loop
                        StartupDone = True
                    End If
                End SyncLock
                'update date
                If RunBox.DateBox.Text <> Format(Now(), "dddd, MMMM d, yyyy")
Then
                    RunBox.DateBox.Text = Format(Now(), "dddd, MMMM d, yyyy")
                End If
                'update time
                If RunBox.TimeBox.Text <> Format(Now(), "HH:mm:ss") Then
                    RunBox.TimeBox.Text = Format(Now(), "HH:mm:ss")
                End If
                'Append the Log Trace Box with any current log text, then clear
the shared variable
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.LogWritten <> "" Then
                        RunBox.LogTrace.AppendText(Windowsill.LogWritten)
                        Windowsill.LogWritten = ""
                        'move the cursor to the end
                        RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LogTrace.ScrollToCaret()
                    End If
                End SyncLock
                'Check and see if the log trace has too many characters
                If RunBox.LogTrace.TextLength > MaxCharacters Then
                    'too many characters in the log trace, eliminate enough
upstream characters to reduce its length to max
                    RunBox.LogTrace.Text = RunBox.LogTrace.Text.Remove(0,
CInt(RunBox.LogTrace.TextLength - MaxCharacters))
                    'move the cursor to the end
                    RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                    RunBox.LogTrace.ScrollToCaret()
                End If
                'Append the Laser Rx Box with any current text, then clear the
shared variable
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.LaserRX <> "" Then
                        RunBox.LaserRxBox.AppendText(Windowsill.LaserRX)
                        Windowsill.LaserRX = ""
                        'move the cursor to the end
                        RunBox.LaserRxBox.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LaserRxBox.ScrollToCaret()
                    End If
                End SyncLock
                'Check and see if the Laser Rx Box has too many characters
                If RunBox.LaserRxBox.TextLength > MaxCharacters Then
                    'too many characters in the laser rx box, eliminate enough
upstream characters to reduce its length to max
                    RunBox.LaserRxBox.Text = RunBox.LaserRxBox.Text.Remove(0,
CInt(RunBox.LaserRxBox.TextLength - MaxCharacters))
                    'move the cursor to the end
                    RunBox.LaserRxBox.Select(RunBox.LaserRxBox.TextLength, 0)
                    RunBox.LaserRxBox.ScrollToCaret()
```

```
                End If
                'check for program exit status
                'bypass startup if shutdown condition exists
                SyncLock CentralClass.ProgramExitLock
                    LocalExit = CentralClass.ExitProgram
                End SyncLock
                If LocalExit = True Then StartupDone = True
                Application.DoEvents()
                Thread.Sleep(55)
            Loop While StartupDone = False

            'Transmit the laser init string
            If ProgramMode <> 0 Then
                'start the thread to transmit the string
                SyncLock Windowsill.RunBoxLock
                    Windowsill.InitComplete = False
                    LaserInitTime = Now.Ticks
                    Windowsill.FirstOrders.Start()
                End SyncLock
            Else
                'indicate that laser init is complete
                SyncLock Windowsill.RunBoxLock
                    Windowsill.InitComplete = True
                End SyncLock
            End If

            'make sure the laser init string completes transmission
            StartupDone = False
            Do
                'check for completion of transmission of the init string
                If ProgramMode <> 0 Then
                    'the string should be transmitted; check on completion
                    SyncLock Windowsill.RunBoxLock
                        If Windowsill.InitComplete = True Then
                            'laser init finished, proceed to main loop
                            StartupDone = True
                        End If
                    End SyncLock
                    If (StartupDone = False) And (Now.Ticks - LaserInitTime >
LaserInitTimeout) Then
                        'too long waiting for the laser to finish init
                        StartupDone = True
                        Lumberjack.SendToLog("Error:  Timeout in Laser
Initialization, Terminating")
                        'abort program
                        SyncLock CentralClass.ProgramExitLock
                            CentralClass.ExitProgram = True
                        End SyncLock
                        LocalExit = True
                    End If
                Else
                    StartupDone = True
                End If
                'update date
                If RunBox.DateBox.Text <> Format(Now(), "dddd, MMMM d, yyyy")
Then
                    RunBox.DateBox.Text = Format(Now(), "dddd, MMMM d, yyyy")
                End If
                'update time
```

```
                If RunBox.TimeBox.Text <> Format(Now(), "HH:mm:ss") Then
                    RunBox.TimeBox.Text = Format(Now(), "HH:mm:ss")
                End If
                'Append the Log Trace Box with any current log text, then clear
the shared variable
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.LogWritten <> "" Then
                        RunBox.LogTrace.AppendText(Windowsill.LogWritten)
                        Windowsill.LogWritten = ""
                        'move the cursor to the end
                        RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LogTrace.ScrollToCaret()
                    End If
                End SyncLock
                'Check and see if the log trace has too many characters
                If RunBox.LogTrace.TextLength > MaxCharacters Then
                    'too many characters in the log trace, eliminate enough
upstream characters to reduce its length to max
                    RunBox.LogTrace.Text = RunBox.LogTrace.Text.Remove(0,
CInt(RunBox.LogTrace.TextLength - MaxCharacters))
                    'move the cursor to the end
                    RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                    RunBox.LogTrace.ScrollToCaret()
                End If
                'Append the Laser Rx Box with any current text, then clear the
shared variable
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.LaserRX <> "" Then
                        RunBox.LaserRxBox.AppendText(Windowsill.LaserRX)
                        Windowsill.LaserRX = ""
                        'move the cursor to the end
                        RunBox.LaserRxBox.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LaserRxBox.ScrollToCaret()
                    End If
                End SyncLock
                'Check and see if the Laser Rx Box has too many characters
                If RunBox.LaserRxBox.TextLength > MaxCharacters Then
                    'too many characters in the laser rx box, eliminate enough
upstream characters to reduce its length to max
                    RunBox.LaserRxBox.Text = RunBox.LaserRxBox.Text.Remove(0,
CInt(RunBox.LaserRxBox.TextLength - MaxCharacters))
                    'move the cursor to the end
                    RunBox.LaserRxBox.Select(RunBox.LaserRxBox.TextLength, 0)
                    RunBox.LaserRxBox.ScrollToCaret()
                End If
                'check for program exit status
                'bypass startup if shutdown condition exists
                SyncLock CentralClass.ProgramExitLock
                    LocalExit = CentralClass.ExitProgram
                End SyncLock
                If LocalExit = True Then StartupDone = True
                Application.DoEvents()
                Thread.Sleep(55)
            Loop While StartupDone = False

            'finish starting up
            'only enter main mode if the program is not shutting down
            If LocalExit = False Then
                'enable applicable controls
```

```
                'first, enable speed controls EXCEPT the 'stop' button
                'it is only applicabe once started
                RunBox.ButtonStopSpeed.Enabled = False
                RunBox.OnCavitationContinue.Enabled = True
                RunBox.OnCavitationStop.Enabled = True
                RunBox.OnCavitationStopAndRestart.Enabled = True
                RunBox.SetSpeedInputBox.Enabled = True
                RunBox.RestartDelayInputBox.Enabled = True
                RunBox.ButtonSetSpeed.Enabled = True
                RunBox.ButtonStartSpeed.Enabled = True

                'next, enable laser controls IF the mode is correct
                If ProgramMode <> 0 Then
                    'laser control enabled, so enable the controls

                    'now enable the controls
                    RunBox.AutomaticPulsingTrue.Enabled = True
                    RunBox.AutomaticPulsingFalse.Enabled = True
                    RunBox.LaserRxBox.Enabled = True
                    RunBox.EstimatedDurationInputBox.Enabled = True
                    RunBox.LaserDelayInputBox.Enabled = True
                    RunBox.ButtonSetLaser.Enabled = True
                    RunBox.ButtonFireLaser.Enabled = True
                    RunBox.ButtonSendLaserCommand.Enabled = True
                End If

                SyncLock Windowsill.RunBoxLock
                    LocalLaserPulsePower = Windowsill.NextLaserPulsePower
                End SyncLock
                'log the default selections
                Lumberjack.SendToLog("Default Control Speed:  " &
RunBox.SetSpeedShowBox.Text)
                Lumberjack.SendToLog("Default Delay Before Restart:  " &
RunBox.RestartDelayShowBox.Text)
                'log default action on cavitation
                If RunBox.OnCavitationContinue.Checked = True Then
                    Lumberjack.SendToLog("Default Action on Cavitation:
Continue")
                ElseIf RunBox.OnCavitationStop.Checked = True Then
                    Lumberjack.SendToLog("Default Action on Cavitation:  Stop")
                ElseIf RunBox.OnCavitationStopAndRestart.Checked = True Then
                    Lumberjack.SendToLog("Default Action on Cavitation:  Stop
and Restart")
                Else
                    'Unknown state
                    Lumberjack.SendToLog("Unknown Default Action on
Cavitation")
                End If
                'log default value of start/stop speed control
                If RunBox.ButtonStartSpeed.Enabled = True Then
                    Lumberjack.SendToLog("Default Speed Control State:
Started")
                Else
                    Lumberjack.SendToLog("Default Speed Control State:
Stopped")
                End If
                'log laser setup if mode <> 0
                If ProgramMode <> 0 Then
                    If RunBox.AutomaticPulsingTrue.Checked = True Then
```

```
                        Lumberjack.SendToLog("Default Laser Pulse Control:
Automatic")
                    ElseIf RunBox.AutomaticPulsingFalse.Checked = True Then
                        Lumberjack.SendToLog("Default Laser Pulse Control:
Manual")
                    Else
                        'unknown state
                        Lumberjack.SendToLog("Unknown Default Laser Pulse
Control")
                    End If
                    Lumberjack.SendToLog("Initial Laser Pulse Power:  " &
Str(LocalLaserPulsePower))
                    Lumberjack.SendToLog("Default Estimated Laser Pulse
Duration:  " & RunBox.EstimatedDurationShowBox.Text)
                    Lumberjack.SendToLog("Default Delay Between Laser Pulses:
" & RunBox.LaserDelayShowBox.Text)
                End If

                'log completion of user interface startup
                RunBox.CurrentStatusBox.Text = "Ready"
                Lumberjack.SendToLog("User Interface Startup Complete; Program
Ready")
                SyncLock Windowsill.RunBoxLock
                    Windowsill.FinishedStartup = True
                End SyncLock

                'main loop
                Do
                    SyncLock CentralClass.ProgramExitLock
                        LocalExit = CentralClass.ExitProgram
                    End SyncLock
                    'update date
                    If RunBox.DateBox.Text <> Format(Now(), "dddd, MMMM d,
yyyy") Then
                        RunBox.DateBox.Text = Format(Now(), "dddd, MMMM d,
yyyy")
                    End If
                    'update time
                    If RunBox.TimeBox.Text <> Format(Now(), "HH:mm:ss") Then
                        RunBox.TimeBox.Text = Format(Now(), "HH:mm:ss")
                    End If
                    'Append the Log Trace Box with any current log text, then
clear the shared variable
                    SyncLock Windowsill.RunBoxLock
                        If Windowsill.LogWritten <> "" Then
                            RunBox.LogTrace.AppendText(Windowsill.LogWritten)
                            Windowsill.LogWritten = ""
                            'move the cursor to the end
                            RunBox.LogTrace.Select(RunBox.LogTrace.TextLength,
0)
                            RunBox.LogTrace.ScrollToCaret()
                        End If
                    End SyncLock
                    'Check and see if the log trace has too many characters
                    If RunBox.LogTrace.TextLength > MaxCharacters Then
                        'too many characters in the log trace, eliminate enough
upstream characters to reduce its length to max
                        RunBox.LogTrace.Text = RunBox.LogTrace.Text.Remove(0,
CInt(RunBox.LogTrace.TextLength - MaxCharacters))
```

```
                        'move the cursor to the end
                        RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LogTrace.ScrollToCaret()
                    End If
                    'Append the Laser Rx Box with any current text, then clear
the shared variable
                    SyncLock Windowsill.RunBoxLock
                        If Windowsill.LaserRX <> "" Then
                            RunBox.LaserRxBox.AppendText(Windowsill.LaserRX)
                            Windowsill.LaserRX = ""
                            'move the cursor to the end

RunBox.LaserRxBox.Select(RunBox.LogTrace.TextLength, 0)
                            RunBox.LaserRxBox.ScrollToCaret()
                        End If
                    End SyncLock
                    'Check and see if the Laser Rx Box has too many characters
                    If RunBox.LaserRxBox.TextLength > MaxCharacters Then
                        'too many characters in the laser rx box, eliminate
enough upstream characters to reduce its length to max
                        RunBox.LaserRxBox.Text =
RunBox.LaserRxBox.Text.Remove(0, CInt(RunBox.LaserRxBox.TextLength -
MaxCharacters))
                        'move the cursor to the end
                        RunBox.LaserRxBox.Select(RunBox.LaserRxBox.TextLength,
0)
                        RunBox.LaserRxBox.ScrollToCaret()
                    End If

                    'Check for updated globals to show in the runbox
                    SyncLock Windowsill.RunBoxLock
                        'see if the speed needs to be stopped
                        LocalStopAction = Windowsill.StopAction
                        'check for fired laser
                        If Windowsill.LaserFireIndicator = True Then
                            'laser was recently fired, show it
                            LastLaserPulseTime = Now.Ticks
                            LocalLaserFired = True
                            'clear the flag
                            Windowsill.LaserFireIndicator = False
                        End If
                        'check for laser firing readiness
                        If ProgramMode > 0 Then
                            'only if there's a laser to fire
                            If Windowsill.AutomaticPulsing = False Then
                                'set it yellow
                                HighlightLaserReady = True
                                ShowFireLaser = False
                            ElseIf Windowsill.ProperSpeed = True Then
                                'speed in range, set to yellow
                                HighlightLaserReady = True
                                ShowFireLaser = True
                            Else
                                'not ready, set to default color
                                HighlightLaserReady = False
                                ShowFireLaser = False
                            End If
                        End If
                        'check for disabling of automatic pulsing
```

```
                        If Windowsill.DisableAutomaticPulsing = True Then
                            'disable automaticpulsing
                            LocalDisablePulsing = True
                            Windowsill.AutomaticPulsing = False
                            Windowsill.DisableAutomaticPulsing = False
                        End If
                        'get speed, speed in range, and cavitation status
                        LocalSpeed = Windowsill.DetectedSpeed
                        SpeedInRange = Windowsill.ProperSpeed
                        DetectedCavitation = Windowsill.Cavitation
                        'get setspeed, restart delay, pulse duration, pulse
delay
                        LocalSetSpeed = Windowsill.DesiredSpeed
                        LocalRestartDelay = Windowsill.RestartDelayTime
                        LocalPulseDuration =
Windowsill.EstimatedLaserPulseDuration
                        LocalPulseDelay = Windowsill.DelayBetweenPulses
                        'get next pulse power
                        LocalLaserPulsePower = Windowsill.NextLaserPulsePower
                        'get run and restart status
                        LocalRestartStatus = Windowsill.RestartStatus
                        LocalRestartTimeLeft = Windowsill.RestartTimeLeft
                        LocalRunButtonClicked = Windowsill.RunButtonClicked
                        LocalCavitationBehavior = Windowsill.CavitationAction
                        'get laser fire sequence stage
                        LocalLaserStage = Windowsill.LaserStage
                    End SyncLock

                    'if necessary, press stop
                    If LocalStopAction = True Then
                        RunBox.PressStopButton()
                        'clear the variable demanding the stop
                        SyncLock Windowsill.RunBoxLock
                            Windowsill.StopAction = False
                        End SyncLock
                    End If

                    If LocalDisablePulsing = True Then
                        'continue to disable autopulsing
                        RunBox.StopAutomaticPulsing()
                        LocalDisablePulsing = False
                    End If

                    'show updated values
                    'update the speed (includes padding/trimming on right hand
side)
                    If RunBox.CurrentSpeedBox.Text <> LocalSpeed.ToString("E2")
Then
                        RunBox.CurrentSpeedBox.Text = LocalSpeed.ToString("E2")
                    End If
                    'update the setspeed
                    If RunBox.SetSpeedShowBox.Text <>
LocalSetSpeed.ToString("E2") Then
                        RunBox.SetSpeedShowBox.Text =
LocalSetSpeed.ToString("E2")
                    End If
                    'update the restart delay
                    If RunBox.RestartDelayShowBox.Text <>
LocalRestartDelay.ToString("E2") Then
```

```
                        RunBox.RestartDelayShowBox.Text =
LocalRestartDelay.ToString("E2")
                    End If
                    'update the pulse duration
                    If RunBox.EstimatedDurationShowBox.Text <>
LocalPulseDuration.ToString("E2") Then
                        RunBox.EstimatedDurationShowBox.Text =
LocalPulseDuration.ToString("E2")
                    End If
                    If FirePulse.DurationDisplay.Text <>
LocalPulseDuration.ToString("E6") Then
                        FirePulse.DurationDisplay.Text =
LocalPulseDuration.ToString("E6")
                    End If
                    'update pulse delay
                    If RunBox.LaserDelayShowBox.Text <>
LocalPulseDelay.ToString("E2") Then
                        RunBox.LaserDelayShowBox.Text =
LocalPulseDelay.ToString("E2")
                    End If
                    'update pulse power
                    If RunBox.NextPulsePowerShowBox.Text <>
LocalLaserPulsePower.ToString("E2") Then
                        RunBox.NextPulsePowerShowBox.Text =
LocalLaserPulsePower.ToString("E2")
                    End If
                    If FirePulse.PowerDisplay.Text <>
LocalLaserPulsePower.ToString("E6") Then
                        FirePulse.PowerDisplay.Text =
LocalLaserPulsePower.ToString("E6")
                    End If

                    'update the indicators
                    If DetectedCavitation = True Then
                        'cavitation has been detected
                        If RunBox.CavitationBox.Text <> "YES" Then
                            'update the box
                            RunBox.CavitationBox.Text = "YES"
                            RunBox.CavitationBox.BackColor = Color.Maroon
                        End If
                    Else
                        'cavitation sensor shows no cavitation
                        If RunBox.CavitationBox.Text <> "NO" Then
                            'update the box
                            RunBox.CavitationBox.Text = "NO"
                            RunBox.CavitationBox.BackColor = Color.Navy
                        End If
                    End If
                    If SpeedInRange = True Then
                        'the speed is in range
                        If RunBox.SpeedInRangeBox.Text <> "YES" Then
                            'update the box
                            RunBox.SpeedInRangeBox.Text = "YES"
                            RunBox.SpeedInRangeBox.BackColor = Color.DarkGreen
                        End If
                    Else
                        If RunBox.SpeedInRangeBox.Text <> "NO" Then
                            'update the box
                            RunBox.SpeedInRangeBox.Text = "NO"
```

```
                        RunBox.SpeedInRangeBox.BackColor = Color.Navy
                    End If
                End If

                'what to show in the Fire Laser! box
                If HighlightLaserReady = True Then
                    'speed in range, set to yellow
                    If RunBox.FireLaserBox.ForeColor <> Color.Yellow Then
                        RunBox.FireLaserBox.ForeColor = Color.Yellow
                    End If
                Else
                    'not ready, set to default color
                    If RunBox.FireLaserBox.ForeColor <>
System.Drawing.SystemColors.ControlLight Then
                        RunBox.FireLaserBox.ForeColor =
System.Drawing.SystemColors.ControlLight
                    End If
                End If
                'show laser fire status
                If LocalLaserFired = True Then
                    'laser has recently been fired
                    'check timeout
                    If (Now.Ticks - LastLaserPulseTime) >
LaserPulseFlashDurationTicks Then
                        'timed out
                        LocalLaserFired = False
                        'clear the flash
                        RunBox.FireLaserBox.BackColor =
System.Drawing.SystemColors.Control
                    Else
                        If RunBox.FireLaserBox.BackColor <> Color.Crimson
Then
                            'set it to flash crimson
                            RunBox.FireLaserBox.BackColor = Color.Crimson
                        End If
                    End If
                End If

                'show the current status
                If LocalRunButtonClicked = True Then
                    'current status is running
                    StatusString = "Running"
                    'append it with other tags
                    'display speed in range
                    If SpeedInRange = True Then
                        StatusString = StatusString & ", Speed in Target
Range"
                    Else
                        StatusString = StatusString & ", Adjusting Speed"
                    End If
                    If LocalRestartStatus = 0 Then
                        'running normally
                        'check for laser readiness
                        If ProgramMode > 0 Then
                            If ShowFireLaser = True Then
                                'laser control active
                                StatusString = StatusString & ", Ready to
Fire Laser"
                            End If
```

```
                            End If
                        ElseIf LocalRestartStatus = 1 Then
                            'stopping
                            StatusString = StatusString & ", Stopping"
                        ElseIf LocalRestartStatus = 2 Then
                            'paused
                            StatusString = StatusString & ", Waiting for
Restart, " & LocalRestartTimeLeft & " s Remaining"
                        End If
                    Else
                        'current status is stopped
                        StatusString = "Not Running, Ready"
                    End If
                    If LocalLaserStage <> 0 Then
                        'reflect the laser firing sequence
                        StatusString = StatusString & ", Laser Pulse at Stage "
& LocalLaserStage.ToString
                    End If
                    'show status text
                    If RunBox.CurrentStatusBox.Text <> StatusString Then
                        RunBox.CurrentStatusBox.Text = StatusString
                    End If

                    'stop if cavitation detected in that mode
                    If DetectedCavitation = True Then
                        'cavitated, check the mode
                        If LocalCavitationBehavior = 1 Then
                            'stop
                            RunBox.PressStopButton()
                        End If
                    End If
                    Application.DoEvents()
                    Thread.Sleep(55)
                Loop While LocalExit = False
            End If

            'shutdown routine

            'disable all controls
            RunBox.ButtonSetSpeed.Enabled = False
            RunBox.ButtonStartSpeed.Enabled = False
            RunBox.ButtonStopSpeed.Enabled = False
            RunBox.ButtonSetLaser.Enabled = False
            RunBox.ButtonFireLaser.Enabled = False
            RunBox.ButtonSendLaserCommand.Enabled = False
            RunBox.ButtonEnd.Enabled = False
            RunBox.ButtonInsertComment.Enabled = False
            RunBox.ButtonHelp.Enabled = False
            RunBox.RestartDelayInputBox.Enabled = False
            RunBox.SetSpeedInputBox.Enabled = False
            RunBox.OnCavitationContinue.Enabled = False
            RunBox.OnCavitationStop.Enabled = False
            RunBox.OnCavitationStopAndRestart.Enabled = False
            RunBox.AutomaticPulsingTrue.Enabled = False
            RunBox.AutomaticPulsingFalse.Enabled = False
            RunBox.EstimatedDurationInputBox.Enabled = False
            RunBox.LaserDelayInputBox.Enabled = False

            'reset colors and status boxes
```

```
            RunBox.CurrentStatusBox.Text = "Shutting Down"
            RunBox.FireLaserBox.BackColor = System.Drawing.SystemColors.Control
            RunBox.FireLaserBox.ForeColor =
System.Drawing.SystemColors.ControlLight
            RunBox.CavitationBox.Text = "NO"
            RunBox.CavitationBox.ForeColor = Color.Yellow
            RunBox.CavitationBox.BackColor = Color.Navy
            RunBox.SpeedInRangeBox.Text = "NO"
            RunBox.SpeedInRangeBox.ForeColor = Color.Yellow
            RunBox.SpeedInRangeBox.BackColor = Color.Navy

            'hide additional boxes
            FirePulse.Hide()
            InsertComment.Hide()
            SendLaserCommand.Hide()
            HelpBox.Hide()

            Application.DoEvents()
            'wait for the right stage
            Do
                'get execution stage
                SyncLock CentralClass.ProgramExitLock
                    LocalExecutionStage = CentralClass.ExecutionStage
                End SyncLock
                'update the speed
                SyncLock Windowsill.RunBoxLock
                    LocalSpeed = Windowsill.DetectedSpeed
                End SyncLock
                If RunBox.CurrentSpeedBox.Text <> LocalSpeed.ToString("E2")
Then
                    RunBox.CurrentSpeedBox.Text = LocalSpeed.ToString("E2")
                End If
                'update date
                If RunBox.DateBox.Text <> Format(Now(), "dddd, MMMM d, yyyy")
Then
                    RunBox.DateBox.Text = Format(Now(), "dddd, MMMM d, yyyy")
                End If
                'update time
                If RunBox.TimeBox.Text <> Format(Now(), "HH:mm:ss") Then
                    RunBox.TimeBox.Text = Format(Now(), "HH:mm:ss")
                End If
                'Append the Log Trace Box with any current log text, then clear
the shared variable
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.LogWritten <> "" Then
                        RunBox.LogTrace.AppendText(Windowsill.LogWritten)
                        Windowsill.LogWritten = ""
                        'move the cursor to the end
                        RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LogTrace.ScrollToCaret()
                    End If
                End SyncLock
                'Check and see if the log trace has too many characters
                If RunBox.LogTrace.TextLength > MaxCharacters Then
                    'too many characters in the log trace, eliminate enough
upstream characters to reduce its length to max
                    RunBox.LogTrace.Text = RunBox.LogTrace.Text.Remove(0,
CInt(RunBox.LogTrace.TextLength - MaxCharacters))
                    'move the cursor to the end
```

```
                    RunBox.LogTrace.Select(RunBox.LogTrace.TextLength, 0)
                    RunBox.LogTrace.ScrollToCaret()
                End If
                'Append the Laser Rx Box with any current text, then clear the
shared variable
                SyncLock Windowsill.RunBoxLock
                    If Windowsill.LaserRX <> "" Then
                        RunBox.LaserRxBox.AppendText(Windowsill.LaserRX)
                        Windowsill.LaserRX = ""
                        'move the cursor to the end
                        RunBox.LaserRxBox.Select(RunBox.LogTrace.TextLength, 0)
                        RunBox.LaserRxBox.ScrollToCaret()
                    End If
                End SyncLock
                'Check and see if the Laser Rx Box has too many characters
                If RunBox.LaserRxBox.TextLength > MaxCharacters Then
                    'too many characters in the laser rx box, eliminate enough
upstream characters to reduce its length to max
                    RunBox.LaserRxBox.Text = RunBox.LaserRxBox.Text.Remove(0,
CInt(RunBox.LaserRxBox.TextLength - MaxCharacters))
                    'move the cursor to the end
                    RunBox.LaserRxBox.Select(RunBox.LaserRxBox.TextLength, 0)
                    RunBox.LaserRxBox.ScrollToCaret()
                End If
                Application.DoEvents()
                Thread.Sleep(55)
            Loop While LocalExecutionStage < 1050

            'Dispose of the forms
            'Dispose the FirePulse box
            Try
                FirePulse.Hide()
                FirePulse.Dispose()
            Catch ex As ObjectDisposedException
            End Try

            'Dispose the Helpbox
            Try
                HelpBox.Hide()
                HelpBox.Dispose()
            Catch ex As ObjectDisposedException
            End Try

            'Dispose the InsertComment box
            Try
                InsertComment.Hide()
                InsertComment.Dispose()
            Catch ex As ObjectDisposedException
            End Try

            'Dispose the Runbox
            Try
                RunBox.Hide()
                RunBox.Dispose()
            Catch ex As ObjectDisposedException
            End Try

            'Dispose the SendLaserCommand box
            Try
```

```
                SendLaserCommand.Hide()
                SendLaserCommand.Dispose()
            Catch ex As ObjectDisposedException
            End Try

            Application.DoEvents()



            'final shutdown steps
            Lumberjack.SendToLog("User Interface Stopping")
            RunBox.Hide()

            'make sure the laser init thread terminates
            SyncLock Windowsill.RunBoxLock
                Try
                    'terminate the laser init thread
                    If Windowsill.FirstOrders.Join(100) = False Then
                        Windowsill.FirstOrders.Abort()
                    End If
                Catch ex As Exception
                    'nothing to do here
                End Try
            End SyncLock

        Catch ThreadNeededKilling As ThreadAbortException
            'the thread was aborted
            Application.ExitThread()
            'make sure the laser init thread terminates
            SyncLock Windowsill.RunBoxLock
                Try
                    'terminate the init thread
                    Windowsill.FirstOrders.Abort()
                Catch ex As Exception
                    'nothing to do here
                End Try
            End SyncLock
        Catch BigException As Exception
            'something got really screwed up
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock

            'make sure the laser init thread terminates
            SyncLock Windowsill.RunBoxLock
                Try
                    'terminate the init thread
                    Windowsill.FirstOrders.Abort()
                Catch ex As Exception
                    'nothing to do here
                End Try
            End SyncLock
        End Try
```

```
    Application.ExitThread()

End Sub



Private Shared Sub LaserInitTX()
    'transmits the initialization commands to the laser, one at a time
    'it is expected that this will be spun off in its own thread

    Dim FullString As String = ""
    Dim LaserInitString(0) As String
    Dim LaserEOL As String = ""
    Dim n As Integer = 0
    Dim TotalCommands As Integer = 0
    Dim WaitTime As Integer = 500
    Dim ProceedStatus As Boolean = False
    Dim LocalClearLine As Boolean = False
    Dim FirstDelay As Boolean = False

    Try

        'set the startup conditions
        Thread.CurrentThread.Name = "FireStarter"
        SyncLock Windowsill.RunBoxLock
            Windowsill.InitComplete = False
            'lock down the laser dataline
            Windowsill.LaserTXDataCritical = True
        End SyncLock

        'get the laser end-of-line character
        SyncLock CentralClass.FileOpsLock
            LaserEOL = CentralClass.LaserNewlineString
        End SyncLock

        'set the laser initialization string as a sequence of commands
        n = 0
        LaserInitString(n) = LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
        LaserInitString(n) = "E=1" & LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
        LaserInitString(n) = "CDRH=1" & LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
        LaserInitString(n) = "CW=1" & LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
        LaserInitString(n) = "P=100" & LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
        LaserInitString(n) = "CW=0" & LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
        LaserInitString(n) = "L=1" & LaserEOL
        n += 1
        ReDim Preserve LaserInitString(n)
```

```
            LaserInitString(n) = "?HID" & LaserEOL
            n += 1
            ReDim Preserve LaserInitString(n)
            LaserInitString(n) = "?S" & LaserEOL
            n += 1
            ReDim Preserve LaserInitString(n)
            LaserInitString(n) = "?INT" & LaserEOL
            n += 1
            ReDim Preserve LaserInitString(n)
            LaserInitString(n) = "?HH" & LaserEOL
            n += 1
            ReDim Preserve LaserInitString(n)
            LaserInitString(n) = "?FL" & LaserEOL
            n += 1
            ReDim Preserve LaserInitString(n)
            LaserInitString(n) = "CAL=4" & LaserEOL


            TotalCommands = UBound(LaserInitString)

            'assemble the full string
            FullString = ""
            For n = 0 To TotalCommands
                FullString = FullString & LaserInitString(n) & vbNewLine
            Next

            'log the transmission
            Lumberjack.SendToLog("Transmitting the Following Laser
Initialization Sequence:  " & FullString)

            Application.DoEvents()

            'transmit the string, one command at a time
            For n = 0 To TotalCommands
                ProceedStatus = False
                FirstDelay = False
                'wait for the line to clear, then xmit command n
                Do
                    If FirstDelay = False Then
                        'inserts a delay from the discovery of a clear line
                        'check the statuses
                        'check for a clear line
                        SyncLock Stevedore.SpeedPortLock
                            LocalClearLine = Stevedore.ClearToFireLaser
                        End SyncLock
                        'then check for an empty buffer
                        If LocalClearLine = True Then
                            SyncLock Windowsill.RunBoxLock
                                If Windowsill.CommandToLaser = "" Then
                                    'good to go
                                    FirstDelay = True
                                End If
                            End SyncLock
                        End If
                        'insert the first delay if ready
                        If FirstDelay = True Then
                            'play nicely, and take a nap
                            Application.DoEvents()
                            Thread.Sleep(WaitTime)
```

```
                            Application.DoEvents()
                        End If
                    Else
                        'passed the first delay, continue
                        'check the line status
                        SyncLock Stevedore.SpeedPortLock
                            LocalClearLine = Stevedore.ClearToFireLaser
                        End SyncLock
                        SyncLock Windowsill.RunBoxLock
                            If Windowsill.CommandToLaser = "" And
LocalClearLine = True Then
                                ProceedStatus = True
                                'transmit
                                Windowsill.CommandToLaser =
Windowsill.CommandToLaser & LaserInitString(n)
                            End If
                        End SyncLock
                    End If
                    'play nicely
                    Application.DoEvents()
                    Thread.Sleep(1)
                    Application.DoEvents()
                Loop While ProceedStatus = False
                'add a second delay post-transmission
                Application.DoEvents()
                Thread.Sleep(WaitTime)
                Application.DoEvents()
            Next

            'finish up
            SyncLock Windowsill.RunBoxLock
                Windowsill.InitComplete = True
                Windowsill.LaserTXDataCritical = False
            End SyncLock

        Catch ThreadNeededKilling As ThreadAbortException
            'the thread was aborted
            Application.ExitThread()
        Catch BigException As Exception
            'something got really screwed up
            Lumberjack.SendToLog("Exception in " & Thread.CurrentThread.Name &
":  " & BigException.Message & vbNewLine & "Details:  " & vbNewLine &
BigException.ToString)
            MsgBox("An exception has occurred in " & Thread.CurrentThread.Name
& " and the program will shut down." & vbNewLine & BigException.Message, ,
"OUCH!")
            SyncLock CentralClass.ProgramExitLock
                CentralClass.ExitProgram = True
            End SyncLock
        End Try

        Application.ExitThread()

    End Sub


End Class
```

B.21 USERGUIDE.TXT

SpeedControl User's Guide

==========
Contents
==========

==========
1. Liability
==========

The User agrees to indemnify the Program Author against any and all damages resulting from any direct or indirect use or misuse of the program. The User shall bear any and all responsibility for the use and/or misuse of the Program.

The Program comes 'As-Is' without any warranty of any kind, express or implied, including suitability for a given purpose. Use of the Program is at the User's own risk.

The Program is *NOT* to be used for, with, or in conjunction with any kind of illicit, unethical, or illegal activity.

==========
2. Introduction
==========

The SpeedContral Program is designed to be used for the control and basic data acquisition of the Centrifugal Tensioned Metastable Fluid Detector (CTMFD) Experiment in the Nuclear Heat Transfer Systems Laboratory at Texas A&M University. It controls the rotational speed of the CTMFD apparatus, issues pulses to the laser system (if connected), and maintains a log of the Program's activities as well as certain sensor readings.

The primary interface of the Program to the Experiment is through the Computer's RS-232 serial port(s). The speed controller box, along with the applicable sensors and a laser pulse line, are connected to a primary serial

port. A laser system, if used, may be connected to the data lines of the primary serial port as well; however, the Program also allows it to be connected through a second RS-232 serial port if the Computer supports it.

The Goal of the Program is fine control of the Experiment's rotational speed and the proper issuance of Laser Pulses in order to find the minimum Laser Pulse Energy that can induce cavitation in the Experiment.

The Program requires a Computer with Windows 2000 or newer, and the .NET Framework 2.0. A modern system with at least two processing cores is highly recommended.

The Laser Control features target a 100 mW Coherent CUBE-series laser system, and may not be compatible with other power levels or systems.

3. Ports and Connections

The Program will use either one (1) or two (2) RS-232 Serial Ports on the Computer.

The Primary Serial Port should have its signal lines connected as follows:

1. GND: Common Signal Ground
2. RTS: Laser Fire Signal
3. DTR: Speed Controller Signal
4. DSR: Speed Sensor Signal
5. DCD: Cavitation Sensor Signal

If a Laser System is used it may be connected to the Primary Serial Port or to a Secondary Serial Port. In either case, its lines should be connected to the port as follows:

1. GND: Common Signal Ground
2. TXD: Data TO Laser Signal
3. RXD: Data FROM Laser Signal

All other lines on the Serial Port(s) are unused.

Aside from the Data (TXD and RXD) and Ground (GND) lines, a Positive (+) signal on the line is used internally as a Boolean 'True' and a Negative (-) signal is interpreted as a Boolean 'False' in the same manner. RS-232 standard voltages apply. The hardware default should be a Negative voltage on every line except Ground.

4. Program Startup & Shutdown

On Startup, a dialog will appear giving the User a choice of Serial Ports found in the system for use as the (Primary) Controller Port and a Laser Port. The User should select (from the list) the Serial Port that is applicable for each connection. For example, if the Speed Controller is connected to COM2, then COM2 should be selected for the Controller port. If a Laser System is

connected to that port as well, then COM2 should also be selected for the Laser Port (it does not need to be connected to the same port). If no laser system is used, then the Laser port should be indicated as 'None' in the list. The proper Port Speed must also be chosen from the list.

The User will also be able to select a Log file for use with the Program. If it exists, it may either be appended or completely overwritten, depending on the selection made by the User. It will be written as a standard text file, so the extension .TXT is highly recommended.

The User may also enable the "Advanced Timing" Feature at the Program's startup; that option will not appear later on. If the checkbox is checked, then the Program's speed measurement will potentially be affected. The Feature only comes into play if it detects chunks of missing execution time in the Program; that can be the result of preemption in a multitasking environment. As the speed is determined by counting changes in an unbufferred serial port control line, such chunks of 'dead time' may have serious effects on the calculated speed's value. If the chunks are large enough, complete cycles of changes will be lost and the speed will be underreported. The Feature combats that by finding chunks of such 'dead time' and excluding them from consideration.

When all the selections are correct and the User is ready, the User should click on the 'OK' button to proceed to the main part of the Program.

The Program may be ended by clicking on 'Exit' in the Startup form or by clicking on 'End' (and confirming the choice to exit) when the main part of the Program is running. All ports and files in use will be closed, and the Program will terminate. The Program may also terminate if it encounters an error.

5. Program Operation

Once the User proceeds to the main part of the Program, a number of options are given in the form. On the top of the main form, there is an information array. This includes the current time, day, speed, log file, and status. It also has color-coded indicators that show whether or not the speed is controlled and within bounds of the desired range as well as a cavitation indicator. Below the information array, speed controls take up space on the left, and laser controls take up space on the right. On the bottom left is a box showing the recently added data to the log file. On the bottom left there are three buttons: Insert Comment, Help, and End. Clicking on End will close the Program, once the User confirms. The Help Button brings up the User's Guide.

The Insert Comment button brings up a window that allows the User to type in a comment and insert it into the log file. Clicking on OK will send whatever is in the box immediately to the log, and will close the window. Clicking on Cancel or the Close Button will close the window without sending anything to the log.

The Speed and Laser controls are discussed in their own sections below.

5.A. Speed Control

There are a number of controls available in the Speed Controls area. On the left, the User may enter the desired speed for the Experiment. The current set speed is shown immediately below the entry box; no changes are made until the User clicks on the 'Set' button in the Speed Controls area.

The User has a choice of three things to do in the event cavitation is detected in the Experiment. The first option is to ignore it and continue running. The second option is to stop (just as if the User clicked on the Stop button while running). The third option, Stop + Restart, halts the Experiment for the amount of time set in the Restart Delay option, and then resumes to the desired speed. The Restart Delay is the amount of time, in seconds, that the Experiment will be held at zero speed until it resumes maintaining the desired speed. No changes to any of those values will be used until the User clicks on the 'Set' button.

The two buttons on the bottom right of the Speed Controls area will start and stop the Experiment. Once the 'Start' button is clicked, it should become unavailable (and 'Stop' will become available), and the Program will begin controlling the speed of the Experiment. Clicking on 'Stop' will end this; it will become unavailable while 'Start' becomes available again, and the Experiment should halt.

The User should be aware of the special values that exist for the set speed. Any valid negative value below -1 will drive the speed control output pin high when the Experiment is Started or Running. This should run the Experiment at its maximum possible speed; however, the speed will never be "In Range" when this feature is used. Values between -1 and 0 will force the fraction of time that the speed control pin is high to the negative of that value; for example, setting it to -0.25 will set the speed control pin to be in the high state 25% of the time. As with values less than or equal to -1, the speed will never be "In Range" for these values. In addition, values for the set speed between 0 and 5 (inclusive) are not considered to have stable resulting speeds. Therefore, in those ranges, the speed will not be considered "In Range" regardless of how close it is to the desired value. It should be noted that automatic laser pulses will not be issued for those speed values as a result of this.

5.B. Laser Control

The Laser Controls area has controls for a laser system, if connected. If the Laser Port was selected as 'None' then the entire area will be unavailable. At the bottom of the area there is a box showing the text received from the laser, if any.

As with the Speed Control area, many of the settings do not take effect until the User clicks on the 'Set' button in the Laser Controls Area. The Estimated Duration and Delay values are set that way. Enabling of automatic pulsing also requires the User to click on the 'Set' button; however, if the User clicks on 'No' for Automatic Pulsing, it is immediately disabled.

The Estimated Duration value is meant to be an estimate of the duration of the laser pulse. It is not measured, and not used for any computations. It is simply stored in the log. However, it will only allow nonzero positive values to be entered.

The Delay value will also only allow positive nonzero values to be entered. It, however, is used by the Program. It defines the minimum time delay (in seconds) between the issuance of automated laser pulses.

The Next Pulse Power value is the laser power demanded of the laser for the next pulse. It will increment automatically when Automatic Pulsing is used. It can be set from the 'Fire Pulse' window.

When Automatic Pulsing is used, the Program will wait for the Experiment's speed to be controlled within range, and will automatically issue pulses to the laser system with the set delay between pulses. Each time, the pulse power will be incremented until one of four things occur: Stopping the Experiment, Disabling Automatic Pulsing, Cavitation, or an out-of bounds pulse power. Automatic Pulsing will be disabled not only if the 'No' button is clicked, but also if the User clicks on the 'Fire Pulse' or 'Send Command to Laser' buttons as well.

The 'Send Command to Laser' button brings up a window that allows the User to enter any text and send it to the laser system. If there is a current operation prohibiting it, the User will be notified and may try to send the command again. An additional button, "Insert EOL" will insert a <CR> character (ASCII code 0xD, 13 in decimal) at the caret location or immediately before the selected text. Without explicit inclusion, no end-of-line characters will be included in the transmission. WARNING -- ANY COMMAND MAY BE SENT TO THE LASER SYSTEM, INCLUDING THOSE THAT HAVE THE POTENTIAL TO CAUSE DAMAGE OR INJURY. YOU CAN **REALLY** SCREW UP AN EXPENSIVE PIECE OF EQUIPMENT OR CAUSE BODILY HARM IF YOU DON'T USE CARE WHEN ISSUING COMMANDS.

The 'Fire Pulse' button brings up a window that allows the User to manually issue a pulse to the laser as well as to modify the values for laser power and estimated pulse duration. As with many other Program settings, these do not take effect until the User clicks on the 'Set Values' button. The 'FIRE' button will *NOT* set the values. The User may also enter a Comment for the log that goes with the manually-issued pulse. Clicking on the 'FIRE' button will disable automatic pulsing and issue the manual pulse.

The Laser Control area has an indicator that reflects the current laser operation. If automatic pulsing is disabled, or, if it is enabled and the speed is in range, it will display 'FIRE LASER' in the indicator. It will flash once a pulse has been issued, whether automatically or manually.

6. Final Remarks

This Program is resource intensive. It is *HIGHLY* recommended that the Computer system have at least TWO CPU cores and is not running anything aside from the Program. It is multithreaded, and one of its threads will use every available CPU cycle it can. This is so that it can read and write to the serial ports thousands of times per second.

Since Windows is a preemptive multitasking system, the speed readout *MAY* be inaccurate. I have not closely examined this. If the thread 'misses' a cyclical change in the speed readout pin, it will not give a fully accurate measurement. A higher-performing system may reduce that vulnerability. The Advanced Timing Feature is designed to alleviate this issue; however, it may introduce its own degree of error.

The Program uses the Win32 high-resolution performance counter. On some systems, this may be erratic. Certain multi-core systems can see issues if the threads get moved from core to core. Other systems may need to disable certain power-saving features if the system's high-resolution performance counter is tied, for example, to a CPU frequency that gets adjusted.

This Program was written in Microsoft Visual Basic 2005 Express.

==========
THE END
==========

APPENDIX C

STAR-CCM+ JAVA MACROS

This Appendix includes the two Java macros used to run the Star-CCM+ simulation and to assist in data extraction.

C.1 GETOUT.JAVA

```
// STAR-CCM+ macro: getout.java
package macro;

import java.util.*;

import star.common.*;
import star.base.neo.*;
import star.vis.*;
import star.base.report.*;

import star.motion.*;
public class getout extends StarMacro {

  public void execute() {
    execute0();
  }

  private void execute0() {

    Simulation simulation_0 =
      getActiveSimulation();

    PointPart pointPart_0 =
      simulation_0.getPartManager().createPointPart(new NeoObjectVector(new
Object[] {}), new DoubleVector(new double[] {0.0, 0.0, 0.0}));

    Region region_0 =
      simulation_0.getRegionManager().getRegion("Fluids");

    pointPart_0.getInputParts().setObjects(region_0);

    PointPart pointPart_1 =
      simulation_0.getPartManager().createPointPart(new NeoObjectVector(new
Object[] {}), new DoubleVector(new double[] {0.0, 0.0, 0.0}));

    pointPart_1.setPresentationName("Copy of point");

    pointPart_1.copyProperties(pointPart_0);

    PointPart pointPart_2 =
      simulation_0.getPartManager().createPointPart(new NeoObjectVector(new
Object[] {}), new DoubleVector(new double[] {0.0, 0.0, 0.0}));

    pointPart_2.setPresentationName("Copy of point");

    pointPart_2.copyProperties(pointPart_0);

    Coordinate coordinate_0 =
      pointPart_1.getPointCoordinate();

    Units units_0 =
```

```
      ((Units) simulation_0.getUnitsManager().getObject("m"));

    coordinate_0.setCoordinate(units_0, units_0, units_0, new DoubleVector(new
double[] {0.0, 0.0, -0.00675}));

    Coordinate coordinate_1 =
      pointPart_2.getPointCoordinate();

    coordinate_1.setCoordinate(units_0, units_0, units_0, new DoubleVector(new
double[] {0.0, 0.0, -0.015}));

    SumReport sumReport_0 =
      simulation_0.getReportManager().createReport(SumReport.class);

    PrimitiveFieldFunction primitiveFieldFunction_0 =
      ((PrimitiveFieldFunction)
simulation_0.getFieldFunctionManager().getFunction("Pressure"));

    sumReport_0.setScalar(primitiveFieldFunction_0);

    sumReport_0.getParts().setObjects(pointPart_1, pointPart_2, pointPart_0);

    ExpressionReport expressionReport_0 =
      ((ExpressionReport)
simulation_0.getReportManager().getReport("CurrentTime"));

    sumReport_0.printReport();

    expressionReport_0.printReport();

    // Get the rotating motion object
    RotatingMotion rotatingMotion_0 =
      ((RotatingMotion)
simulation_0.get(MotionManager.class).getObject("Rotation in Fluids"));

    simulation_0.println("omega = " +
rotatingMotion_0.getRotationRate().getValue());
  }
}
```

C.2 RAMP_UP.JAVA

```
// STAR-CCM+ macro: ramp_up.java
package macro;

import java.util.*;

import star.common.*;
import star.base.neo.*;
import star.base.report.*;
import star.motion.*;

public class ramp_up extends StarMacro {

  public void execute() {
    execute0();
  }
```

```
  private void execute0() {

    // This macro is meant to adjust the rotation rate as a function of time

    // constants
    double pi_val = 3.141592653589793238;
    double rate_min = 0; // Minimum rotation rate, rad/s
    double rate_max = 180 * 2 * pi_val; // Maximum rotation rate, rad/s
    double ramp_begin_time = 0.2; // time to begin ramping
    double ramp_end_time = 2; // time to end ramping
    double sim_time_now = 0; // will be used later for the current time
    double ramp_rate = 0; // will be used later for the ramp rate
    boolean stop_crit_met = false; // stopping criteria

    // Get the current simulation
    Simulation simulation_0 =
      getActiveSimulation();

    // Get the rotating motion object
    RotatingMotion rotatingMotion_0 =
      ((RotatingMotion)
simulation_0.get(MotionManager.class).getObject("Rotation in Fluids"));

    // Get the stopping criteria
    PhysicalTimeStoppingCriterion PhyTimeSC =
      ((PhysicalTimeStoppingCriterion)
simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion("Ma
ximum Physical Time"));

    StepStoppingCriterion StepSC =
      ((StepStoppingCriterion)
simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion("Ma
ximum Steps"));

    AbortFileStoppingCriterion AbortFileSC =
      ((AbortFileStoppingCriterion)
simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion("St
op File"));

    // Check the status of the stopping criteria
    stop_crit_met =
      (AbortFileSC.getIsUsed() && AbortFileSC.getIsSatisfied()) ||
(StepSC.getIsUsed() && StepSC.getIsSatisfied()) || (PhyTimeSC.getIsUsed() &&
PhyTimeSC.getIsSatisfied());

    // Connect to the time inside the simulation
    ExpressionReport TimeVal =
      ((ExpressionReport)
simulation_0.getReportManager().getReport("CurrentTime"));



    while (stop_crit_met == false) {

      // Get the current time
      sim_time_now =
        TimeVal.monitoredValue();
```

```
      // Calculate the new ramp rate
      if (sim_time_now < ramp_begin_time){
        // before the beginning of the ramp
        ramp_rate = rate_min;
        simulation_0.println("Currently Before the Ramp");
      } else if (sim_time_now > ramp_end_time){
        // after the end of the ramp
        ramp_rate = rate_max;
        simulation_0.println("Now After the Ramp");
      } else {
        // during the ramp, ramp linearly
        ramp_rate = ((sim_time_now - ramp_begin_time) * (rate_max - rate_min) /
(ramp_end_time - ramp_begin_time)) + rate_min;
        simulation_0.println("Currently During the Ramp");
      }

      // Set the new rotation rate
      simulation_0.println("Setting the Rotational Rate (rad/s) = " +
ramp_rate);
      rotatingMotion_0.getRotationRate().setValue(ramp_rate);

      // step one time step
      simulation_0.getSimulationIterator().step(1, true);

      // Check the status of the stopping criteria
      stop_crit_met =
        (AbortFileSC.getIsUsed() && AbortFileSC.getIsSatisfied()) ||
(StepSC.getIsUsed() && StepSC.getIsSatisfied()) || (PhyTimeSC.getIsUsed() &&
PhyTimeSC.getIsSatisfied());

    }

  }
}
```