Title

New Hardware and Software Design of a Field-Deployable Real-Time Laser-Based Non-Intrusive Detection System for Measurement of True Travel Time on the Highway

Authors

Cheng, Harry H.
Shaw, Ben
Palen, Joe
et al.

Publication Date

2001-06-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

New Hardware and Software Design of a Field-Deployable Real-Time Laser-Based Non-Intrusive Detection System for Measurement of True Travel Time on the Highway

Harry H. Cheng, Ben Shaw, Joe Palen, Bin Lin, Xudong Hu, Bo Chen, Jason Parks
University of California, Davis

California PATH Research Report
UCB-ITS-PRR-2001-15

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Report for MOU 3005

June 2001

ISSN 1055-1425

Technical Report TR-IEL-2000-102

New Hardware and Software Design of a Field-Deployable Real-Time Laser-Based Non-Intrusive Detection System for Measurement of True Travel Time on the Highway[1]

Harry H. Cheng
Ben Shaw
Joe Palen
Bin Lin
Xudong Hu
Bo Chen
Jason Parks
Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California, Davis
Davis, CA 95616

December 30, 2000

[1] The project is funded by the Caltrans through the California PATH Program

Contents

Abstract

A new version of a field-deployable real-time laser-based detection system has been developed using new techniques of electronics and optics. The laser-based non-intrusive detection system uses a laser line that is projected onto the ground as a probe. The reflected light is collected and focused into a photodiode array by an optical system. Vehicle presence is detected based on the absence of reflected laser light. By placing two identical laser/sensor pairs at a known distance apart, the speeds of both the front and rear of a vehicle are measured based on the times when each sensor is triggered. The length of each vehicle is determined by using these speed measurements and the residence time of the vehicle under each sensor. A new version field prototype of a detection system for real-time measurement of delineations of moving vehicles for highway testing has been developed based on our previous research on the laboratory prototype of the system. The new version utilizes new techniques in electronics, optics and software. The testing results show that the new system can obtain the accuracy of measurement necessary to distinguish moving vehicles on the highway. It is an important step to approach the final goal of this project, described in the last report as developing a roadway detection system that can be used to gather reliable travel time data non-intrusively. The new system quantitatively proved that the principle of measurement is feasible, instead of qualitatively as did the last version system. Several tests have been done with the field prototype system on the highway. The software for real-time data acquisition, data processing and graphical user interface has been developed in the Windows NT system with RTX real-time extension. In the software, the speed, acceleration, and length of a detected vehicle can be calculated and displayed simultaneously. This document describes the design and implementation of each functional component and software of the new field prototype system. The measurement and calculation of laser power has been performed to ensure that the system is safe to expose to the public during field tests and future operation on the highway.

Chapter 1 Introduction

Both CalTrans and the US Department of Transportation have come to the conclusion that it is impossible to build our way out of traffic congestion. The solution is to run the transportation system more intelligently, which is known as the "Intelligent Transportation System" or ITS. Travel time is the most important aspect of the Intelligent Transportation System (ITS). Travel time is a good indicator of other direct constraints on ITS efficiency: cost, risk, and attentive workload. The importance of travel time is verified in ATIS user surveys which indicate that what travelers most want from a transportation system is (almost always) reduced travel time and higher reliability (e.g. reduced travel time variance and reduced risk) [1]. Every traveler must implicitly or explicitly make an assessment of these various travel time options before embarking on every trip; therefore this information is definitely of high value. Because trip travel time is the parameter the public most wants to minimize, this is the parameter that is most important for transportation service providers to measure and minimize.

Speed is commonly used as an indicator of the travel time across a link. In current practice, speed is measured at one or more points along a link and extrapolated across the rest of the link [1]. This extrapolation method is used regardless of the mechanism of detection. Example detection methods are loops---which determine speed from two elements twenty feet apart; radar---which can directly determine speed from the carrier frequency shift (Doppler effect); or video image processing---which tracks vehicles across the pixel elements within the field of view. The extrapolation from a point to a line is not necessarily valid. At the onset of flow breakdown, the speed variations along the length of a link can be quite large. Also, the onset of flow breakdown is when routing decisions are most time-critical and accurate information has the highest value, so inaccurate extrapolations could have detrimental effects to the traveler.

An alternate method to determine the traverse travel time (e.g. the true link speed) is to use Vehicles As Probes (VAP). A VAP system determines travel time directly by identifying vehicles at the start of the link and re-identifying them at the end of the link, with the time difference being the true travel time. The problem with VAP systems is that they require large numbers of both vehicle tags and tag readers to be effective, and the cost justification of such a system seems unwarranted in the light of other options. The key aspect to measuring the actual travel time is simply to identify some distinguishing characteristic on a vehicle at the beginning of a link and then to re-identify that same characteristic on the same vehicle at the end on the link. This is the basic idea of VAP, however the characteristic does not have to be entirely unique (as in a vehicle tag), and it does not necessitate the infrastructure set-up costs of VAP. If a characteristic can be found to separate the fleet into (say) 100 classifications, ``the maximum probability fit'' can be determined for the same sequence of classifications at the downstream detector as was identified at the upstream detector. This is what is currently being done in Germany with the low-resolution imaging provided by (new high speed) loops [1]. If a higher-resolution detector is used so that it is possible to get a few thousand classes, then it should be quite possible to perform a 100% upstream-downstream Origin and Destination (O/D) analysis (even if a significant percentage of the vehicles switch lanes) using time gating and other relatively straight-forward signal processing techniques. The mechanism of detection must allow highly resolved delineations between commonly available "commuter" vehicles, because commuter vehicles represent the majority of the vehicle stream during the period that travel time information is most needed (e.g. the peak hours).

Any mechanism to measure travel time, by definition, is only determining the "past state" of the transportation system. Collecting data on what happened in the past has no utility except if it is used to infer what may happen in the future. All decisions, by definition, are based on an inference of future consequences. When a traveler learns that speed on a route is 50 MPH, the traveler generally infers that the speed will remain 50 MPH when she/he traverses it. This may or may not be an accurate inference. Travelers want to know the

"state" of the system (in the future) when they traverse it. In the simplest case, this is just a straight extrapolation of current "state". More sophisticated travelers may develop their own internal conceptual model of the typical build up and progression of congestion along routes with which they are familiar. A major benefit of ITS will be to provide travelers with a much more valid and comprehensive "look ahead" model of the (short-term) future state of the transportation system. Validation of any traffic model requires (either implicitly or explicitly) traffic O/D data. The lack of valid O/D data has been the major impediment in the calibration, validation and usage of traffic models. In this research project we are developing a roadway detection system that can directly determine O/D data non-intrusively without violating the public's privacy (as in license plate recognition systems).

1.1 Comparison of Different Detection Schemes

Our detection system has a number of advantages over other systems currently in use. In current practice, vehicle features are most commonly measured using inductive loops or video image processing. An advantage of our system over loop detectors is the relative ease of installation and maintenance. Because loops are buried beneath the pavement, installation requires heavy equipment, and traffic must be re-routed [2]. It is for this reason that loops are expensive to install and repair. Because our system is mounted above the road, once installed, it can be maintained without disrupting the flow of traffic. More importantly, loop detectors cannot be relied upon to produce accurate speed (and therefore length) measurements because the inductive properties of the loop and loop detectors vary [2]. Video can be used to directly measure the length of vehicles. However, the use of real time video image processing is problematic due to its computationally intensive nature. Our system operates on a simple "on/off" basis, requiring much less computation for vehicle detection, and consequently much less computational hardware. Because video is a passive system (gathering ambient light), video images are dependent on the lighting conditions. Vehicle length measurements taken from video, even on the same vehicle, may not produce consistent results depending on time of day and weather conditions. For truly site and time independent vehicle length measurements, video would require an external source of illumination. Because our system is active, it produces its own signals to be sensed and it does not suffer from these limitations.

One system that bears some similarity to the system we have developed is the Automatic Vehicle Dimension Measurement System (AVDMS) developed by the University of Victoria [3]. The AVDMS uses laser time-of-flight data to classify vehicles based on length, width, or height, and is based on the Schwartz Electro-Optics Autosense III sensor [4][5][6][7]. The Schwartz systems are entirely dependent on time-of-flight laser measurements with moving parts, similar to conventional lidar (laser radar) in the principle of measurement. There are some significant functional differences between our system and Schwartz's. For example, the fundamental mechanism of detection is that the Schwartz detector determines the range (or distance) from the detector to the objects being detected. Our detector functionally does not determine the range (or distance) from the detector to the objects being detected. The laser of the Schwartz's detector reflects off the vehicle to determine the size, shape, and "presence" of the vehicle. In our detector, the laser reflects off the pavement. The lack of a reflection determines the size, shape, and "presence" of the vehicle. Therefore, our system will be more reliable because of its simplicity.

In comparison with other conventional traffic detection techniques, our system will offer the following salient features:

- The system is mounted above the road and is relatively easy to install. Traffic need not be rerouted.
- The system is insensitive to ambient lighting conditions due to the active lighting source (the laser). It detects every passing object more than 46 cm (18 in) tall in all lighting conditions. No vehicles are missed, yielding nearly 100% accuracy.

- The laser and detector have no moving parts, giving the system high reliability. The primary raw data gathered by the sensor are computationally easy to process.
- Not only does the detector produce local vehicle speed, vehicle volume, and vehicle classifications, but it also allows highly deterministic re-identification of vehicles between sites, even under high flow conditions. Point-to-point travel time, incident detection, and Origin/Destination data can easily be determined with this detector.
- The system has very low power and communication bandwidth requirements, allowing the development of a stand-alone detector untethered from hard-wired infrastructure.

Therefore, our laser-based detector has much higher resolution and is much more accurate than conventional traffic detectors, allowing the determination of point-to-point travel time which currently deployed traffic detection systems do not generate.

1.2 Previous Work

Laboratory and field prototypes were developed to verify the principle the detection system in the previous project. The detection system hardware was constructed using off-the-shelf parts for rapid assembly and easy modification [12]. The laser chosen for use in the indoor prototype system was a Melles Griot model 56 DIL 452/P1 laser line projector. This product has a 30 mW red laser diode and all of the necessary optics for line projection built into a single package. In the laboratory prototype system, the current output from the sensor is amplified and conditioned for interfacing with a Delta Tau PMAC digital I/O board. The data are then collected in real-time by a computer for processing. Sensor software performs low-level gathering of the detector data and an application programmers interface (API) provides a standard way of accessing the data. Applications use this sensor API to display the data in various ways. The X window program shows the time history of the sensor data, displaying multiple detector values at regular intervals and profiles of vehicles as they pass under the sensor. The text based interface displays raw data. The laboratory and field prototype was tested in the laboratory and on the roof of a building in a simulation environment and on the highway in real traffic. The experimental results showed that the principle of the detection was feasible. But the accuracy of measurement was not high enough to distinguish the moving vehicles on the highway, because of their low optical resolution and slow time response.

1.3 System View of the Field Prototype

In the field prototype system, vehicle length is used as the primary identifying feature and is measured using two laser-based vehicle detectors. The system operates in the following manner, as illustrated in

Figure 1.1. The basic detector unit consists of a laser and a spatially offset photodetector positioned above the plane of detection. The laser is a pulsed infrared diode laser that utilizes line-generating optics, which project to a flat planar surface where objects are to be detected. The detector consists of imaging optics and a linear photodiode array. The offset photodiode array receives the laser light that is reflected back from the region of detection. The signal from the photodiode is amplified and sent to a computer for processing. Vehicle presence is detected based on the absence of reflected laser light. Two of these units are integrated and placed a known distance apart, allowing the velocity of the object and its residence time under each detector to be measured, giving the object's length and top-down outline profile.

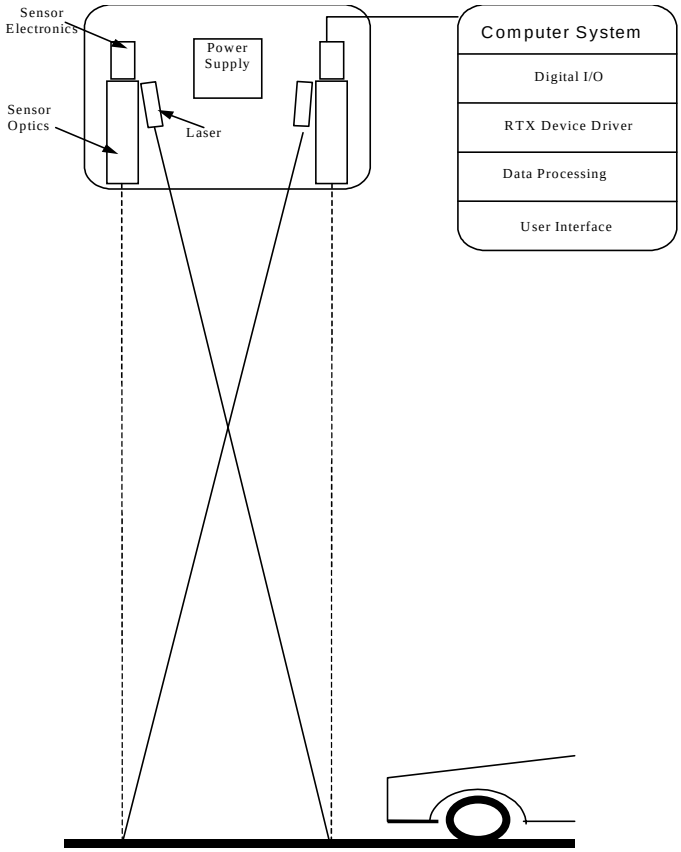


Figure 1.1: System Overview

Figure 1.2 shows the positioning of the system hardware. The detector is mounted at a distance of about 6.4m (21 ft) (the height of a typical highway overpass) above the highway. The distance between each component of a laser/sensor pair is 30.5 cm (1.00 ft). The offset between the two sensor pairs is 10 cm (4 in). The sensors are mounted in a fixed vertical position, pointing downward, and are focused on the ground, forming two detection zones. The lasers are pointed towards the detection zones and are mounted at an adjustable angle, allowing the system to be mounted at different heights. The detection zones stretch across the width of the lane and are each about 13 mm (0.5 in) wide in the direction of traffic flow. In this configuration the minimum detectable object height, also called the critical height, is about 46 cm (18 in). This is lower than the bumper height of most common vehicles. For objects below this height, the laser line will still be visible by the sensor. This can result in the object remaining undetected or can cause signal spike due to reflections, depending on the surface properties and geometry or the object. In either case, for vehicle bumper heights below the critical height, the speed and length measurements will be incorrect due to the fact that one or more of the vehicle edges will be incorrectly found.

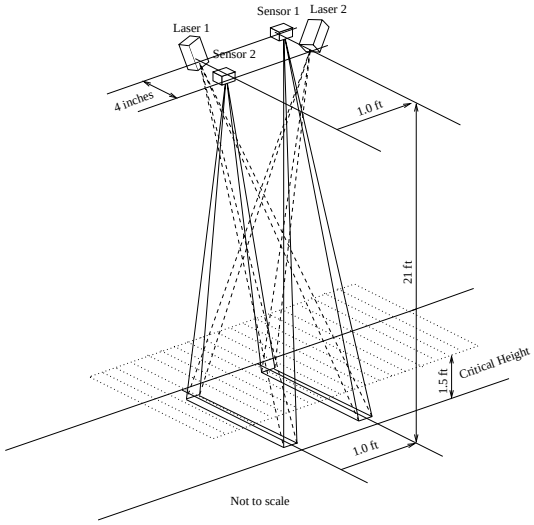


Figure 1.2: System Hardware Configuration

When a vehicle moves into a detection zone, it blocks the laser from being received by the sensor. When the first beam is blocked the current time is recorded. When the second beam is blocked, a second time is recorded. These times give the speed of the front of the car. In a similar manner, when each of the beams is no longer blocked, the times are recorded and the speed of the rear of the vehicle can be calculated. The time that each detector is blocked is also recorded and is used to calculate the vehicle length, assuming constant vehicle acceleration. A more detailed description of the speed and length measurement algorithms is presented in the software section. The assumption of constant acceleration is valid for free-flow traffic conditions, where there is negligible acceleration, and for conditions where the vehicle is accelerating or decelerating uniformly during the time it is in the detection zone. These cover the majority of situations, but there are a few situations, such as stop-and-go traffic, where this basic detection method will not work well.

Chapter 2 Field Prototype system

2.1 Laser and Optics

2.1.1 Laser

Two off-the-shelf integrated diode laser systems, ML20A15-L2 high power diode laser systems from Power Technology Inc are used as the laser source in the current prototype. This is an integrated laser system that incorporates a DC/DC voltage converter, voltage regulator, pulse generator, laser diode and line generation optics into a single unit. Operation requires only a +9V - +14.5V DC power supply and a trigger pulse. The system has a peak power output of 20W at 905nm, with a pulse width of 15ns. It can be pulsed at a maximum rate up to 10 kHz. The line generating optics produces a beam with a full fan angle of 15 degrees. Its high performance and small size make it a good candidate for use in the field deployable prototype system.

A wavelength of 905nm for the laser was chosen for a number of reasons. Infrared light has good transmittance through fog, giving the system better performance under a larger range of weather conditions. Furthermore, the intensity of sunlight around the wavelength of the laser is a local minimum, giving the system better rejection of noise due to sunlight. An infrared laser was also thought to be more appropriate for outdoor use because it is invisible to the human eye, and would therefore cause no distraction to passing motorists.

2.1.1.1 Laser safety

In the field-deployable prototype it is necessary that the laser be eye-safe. The ML20A15-L2 has a pulse width of 15ns, a maximum output power of ~20 W and a maximum frequency of 10 kHz. With the assumptions that the laser is located 6 m above the roadway, the laser line is 4 m long and 5 mm wide, and that a potential observer is 2 m above the roadway as shown in Figure 2.1, the safety of the laser is verified in this section.

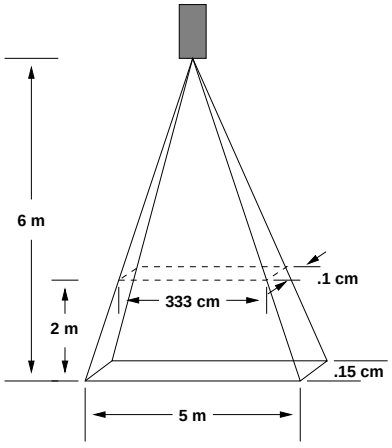


Figure 2.1 Projection Area of the Laser

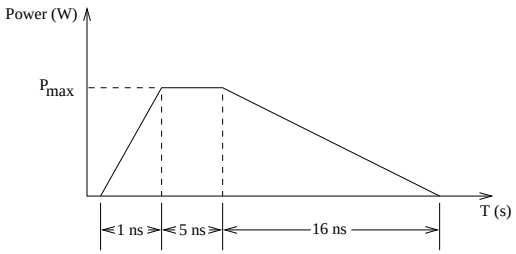


Figure 2.2: Laser Pulse Waveform

Assuming the laser pulse has the form shown in Figure 2.2 and the area of exposure is 267 cm x 0.33cm, the radiant exposure per pulse of the laser system can be calculated by:

$$\frac{H_{\exp osure}}{pulse} = \frac{(\frac{1}{2} \cdot 5ns + \frac{1}{2} \cdot 16ns) \cdot 20W}{L \cdot W} = 5.33 \times 10^{-9} \, J/cm^2$$

For the pulse width of 32×10^{-9}s, the single pulse maximum permissible exposure (MPE) for a 905nm laser expressed as radiant exposure (H) is given by [10]:

$$MPE : H = 5 \cdot C_A \times 10^{-7} \, J/cm^2$$

With C_A = 2.56 for our laser, this becomes

$$H = 1.28 \times 10^{-6} \, J/cm^2$$

The MPE per pulse for repetitively pulsed intra-beam viewing is $n^{-1/4}$ times the MPE for a single pulse exposure where n is the number of pulses found from the viewing time ($T_{viewing}$) and the pulse frequency (f_{pulse}). Assuming a pulse frequency of 10 kHz, the maximum safe viewing time can be found by equating the MPE per pulse and the radiant exposure per pulse.

$$\frac{MPE}{pulse} : H = \frac{H_{\exp osure}}{pulse}$$

$$n^{-1/4} \cdot 1.28 \times 10^{-6} \, J/cm^2 = 5.33 \times 10^{-9} \, J/cm^2$$

$$n = 3.32 \times 10^{-9}$$

$$T_{viewing} \cdot f_{pulse} = 3.32 \times 10^{-9}$$

$$T_{viewing} = 3.32 \times 10^{-5} s \approx 92h$$

The laser is safe for viewing times up to 92 hours. Although it is unlikely that a driver would stop on the highway and stare into the beam for this long, warning signs will be posted to prevent it.

In the June 1999, we measured the laser power of our prototype detection system at both UC Berkeley and UC Davis, and performed extensive calculations using computer software for laser safety with the laser safety officer at UC Davis. The calculation results indicate that the laser power of our detection system is below the minimum national laser safety standard by a large margin.

2.1.1.2 Laser Spectrum and Filter

A bandpass filter that is matched with the wavelength of the laser is used to reduce the level of ambient light received by the sensor. The wavelength of the most of diode lasers varies with temperature in the rate of 0.2-0.5nm/C°. When a filter is used to eliminate the ambient light, it is important to ensure that the wavelength of laser is within the window of the band-pass filter. According to the specifications of the manufacture, the wavelength range of the laser diodes we used is 905±5nm at room temperature. In order to operate our system in wide temperature range, i.e. in the hot summer and cold winter, the window of filter should cover the whole range of laser wavelength when temperature changing. The wavelength does not change too much when the temperature goes lower than room temperature, but it changes a lot at high temperature. The characteristics of a laser spectrum varies for lasers from different manufacturers and manufactured at different times, so it is necessary to measure the real spectrum of laser diode before the filer is chosen. In our laboratory the laser spectrum was measured by a portable fiber optic spectrometer from Ocean Optics, Inc. The laser was guided into the spectrometer by an optical fiber. The measurements were carried out for temperature of 25-47C°. The calibration error is shown in Table 1. The tolerance of calibration is less than 0.6nm.

True Wavelength	Pixel	Pixel ^{2}	Pixel ^{3}	Predicted Wavelength	Difference
922.45	796	633616	504358336	922.447848	0.002152
912.3	775	600625	465484375	912.535575	-0.235575
852.14	650	422500	274625000	852.1652	-0.0252
842.46	629	395641	248858189	841.793027	0.666973
826.45	598	357604	213847192	826.360824	0.089176
811.53	568	322624	183250432	811.289184	0.240816
800.62	548	300304	164566592	801.166424	-0.546424
794.82	536	287296	153990656	795.063968	-0.243968
772.4	492	242064	119095488	772.503496	-0.103496
763.51	475	225625	107171875	763.709175	-0.199175
750.39	450	202500	91125000	750.6976	-0.3076
738.4	426	181476	77308776	738.118288	0.281712
727.29	405	164025	66430125	727.040515	0.249485

Table 1: Calibration Results of Spectrometer in infrared Range

Figure 2.3 shows a typical spectrum of diode laser with an advertised wavelength of 905nm; the actual peak (central) wavelength was about 916nm. The spectra were also measured with two different filters, as shown in Figure 2.4. The filters A and B are 904-DF-15 and 904-DF-30 from Omega Optical, Inc., respectively. The central wavelength increased at the rate of 0.3nm/C° when the temperature increased. The central wavelength was almost unchanged when the temperature was lower than 25C°. In Figure 2.4, the central wavelength with the filter is lower than without the filters. This is due to that the filters blocked the higher part of wavelength. We can see from this figure that the tolerance of central wavelength and half bandwidth is big and the filter can block some of laser when temperature going up. In order to overcome this problem, two new filters were ordered. Two 910-DF-30 filters with central wavelength of 910nm and half bandwidth of 30nm were chosen according to the measurement of laser spectrum. The filters were tested by the manufacturer before they were shipped to ensure the proper central wavelength and half bandwidth.

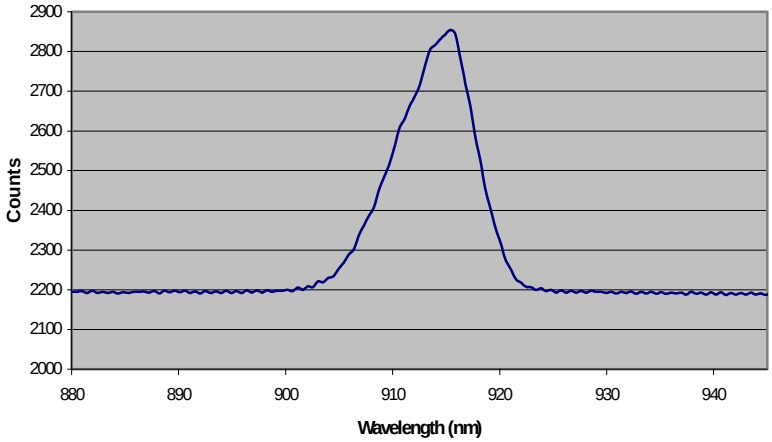


Figure 2.3: Typical Spectrum of Diode Laser

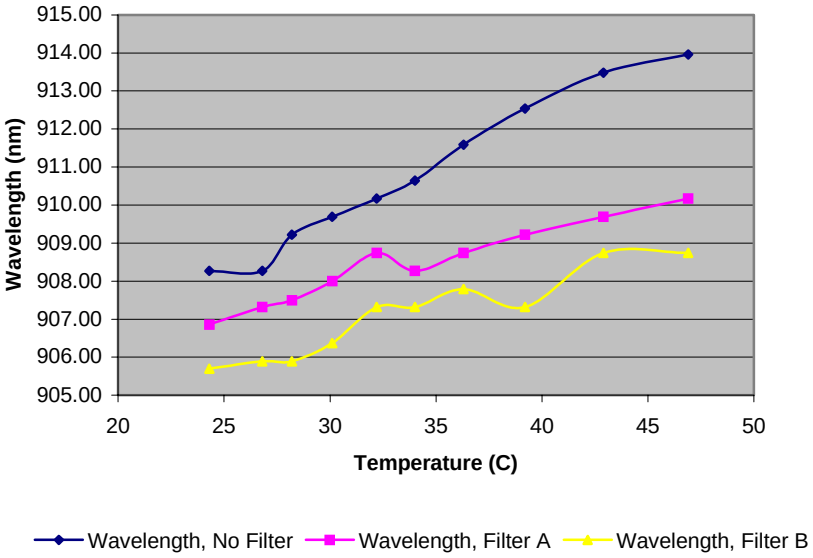


Figure 2.4: Central Wavelength of Diode Laser vs. Temperature

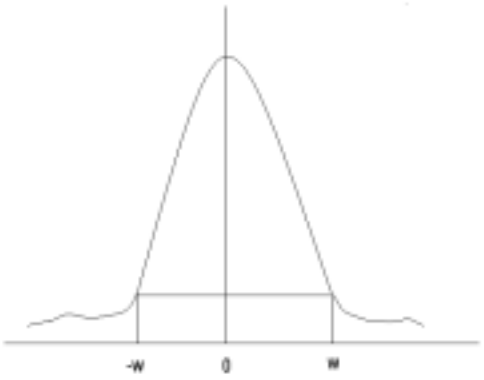


Figure 2.5: Typical profile of a diode laser

2.1.1.3 Profile of Laser Beam

A diode laser is a multi-mode emitter and sometimes the beam profile is not a real Gaussian distribution, as shown in Figure 2.5. The setup shown in Figure 2.6 was designed to verify this analysis. We placed a reflector A, which has good diffusive reflectance, in the field of view of sensor. The position of the reflector is much higher than the critical height, but does not block the laser. In this case, we can get the reflection signal of the sensor in the same side of the laser. This reflection signal is comparable to the signal reflected from road (or the wall in the lab). When we move the laser closer to the reflector, we get higher reflection. Figure 2.7 shows the relationship between amplitude of reflection signal and *d*, distance between reflective material and laser beam. Figure 2.8 indicates the amplitude of reflection signal vs. *H* , the distance of the reflector from the detection system. In the highway situation it is possible the reflectance of some vehicles is higher than that of the reflector we are using in the lab.

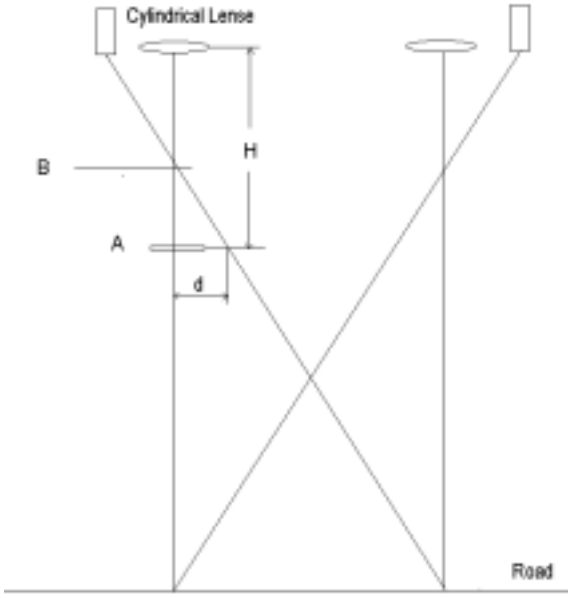


Figure 2.6: Experimental Setup for Side-laser Test

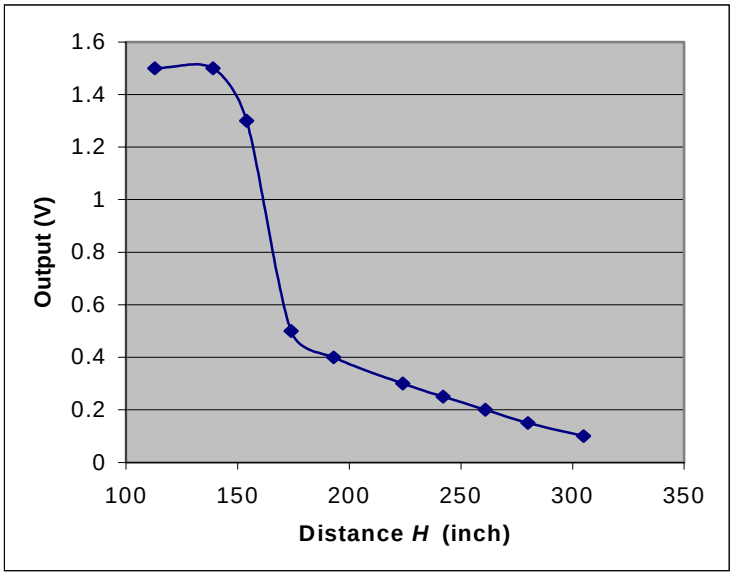


Figure 2.7: Amplitude of reflection as a function of distance between laser and reflector

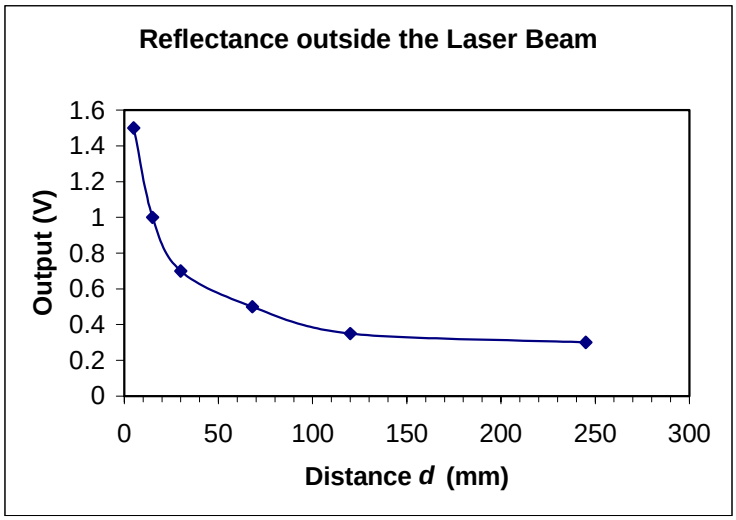


Figure 2.8: Amplitude of reflection vs. distance between reflector and detection system

The solution for reflection from the higher order beam is to place a slot a certain distance from the opening of laser. The slot will block the higher-order beam as long as the distance is suitable. The distance depends on the beam shape and mode. Fortunately, for our laser we can place the slot about 8 inches away from the laser and it blocks 90% of the side beam. Figure 2.9 shows the new system configuration we have designed. There are some diffractive light and/or high-order peaks outside the width of Gaussian profile. This diffractive side beam is hard to see, but we observed a high order laser beam beside the main laser beam by using an IR detection card. This side laser beam could be much lower than the laser peak. If the vehicle has high reflectance, we can still get significant reflection from this portion of beam. This reflection could be comparable to the reflection from the road.

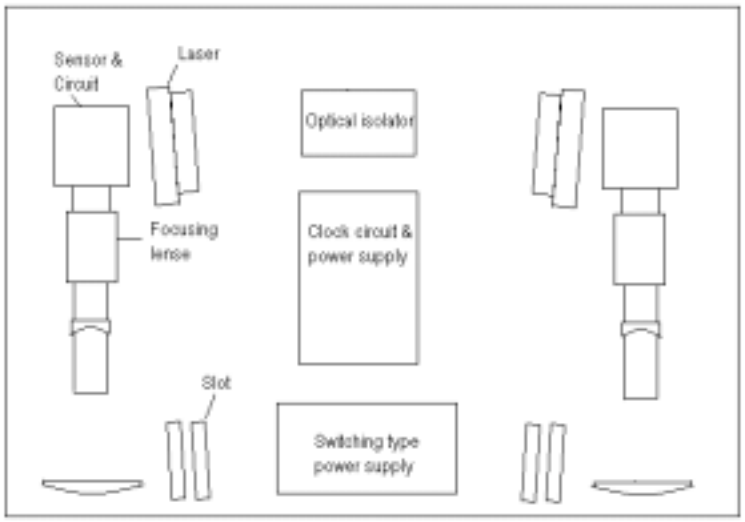


Figure 2.9: Configuration of System for Side-Laser Blocking

2.1.2 Sensor Optics

The sensor optics consists of an imaging lens system and a telescopic lens system. The imaging lens system focuses the reflected laser light onto the active area of the sensor array. The imaging lens was selected based on the criteria that it should have an adjustable focal length within a range around the desired focal length, and that it should have a field-of-view large enough to capture the width of an entire lane and that it should be compact for easy integration into the outdoor system.

Based on the assumptions that the lane width (h_o) is around 3.05 m (10.0 ft) and that the unit will be mounted about 6.40 m (21.0 ft) above the roadway (s_o) and given that the sensor is 7.5 mm (0.295 in) long (h_i), an image distance (s_i) was calculated for the sensor using Equation 2, where it was determined that s_i = 15.8 mm (0.620 in).

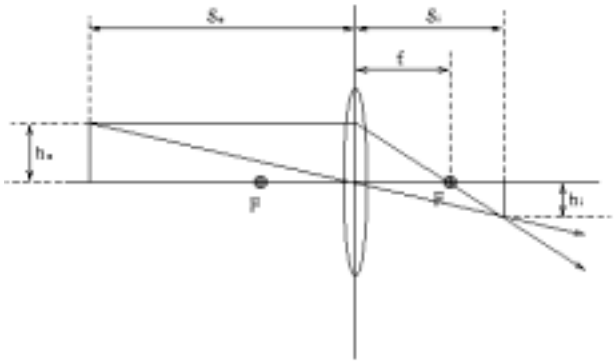


Figure 2.10: Lens Parameter

$$s_i = s_o \frac{h_i}{h_o} \qquad (2\text{-}1)$$

The desired focal length (f) of the lens was then calculated using Equation (2-2)

$$f = \frac{1}{\frac{1}{s_i} + \frac{1}{s_o}} \qquad (2\text{-}2)$$

The focal length was calculated to be 15.7 mm (0.616 in). As a practical matter, the sensor array is placed at the focal point of the imaging lens system. Because s_o is large in comparison with s_i, f is nearly equal to s_i.

The lens we selected, a Tamron 23VM816, has an adjustable focal length of between 0.315 in (8 mm) and 0.630 in (16 mm) and was selected because of this feature. The Tamron lens is also suitably compact and has a field of view that should be large enough to capture the entire lane width. Any lens system that has the correct focal length and an acceptable field-of-view could be used.

The telescopic lens system is mounted in front of the imaging lens system. It is designed to restrict the field-of-view of the imaging lens along the width of the laser line, but not alter the field-of-view along the length of the line. Because the laser line is much longer than it is wide, use of the imaging lens alone would result in a much wider strip of pavement being visible to the sensor than is desired. The telescopic lens system is used to match the dimensions of the laser line image with those of the sensor array. Figure 2.10 shows the imaging lens and the cylindrical lenses used in our system.

The telescopic lens system consists of one positive plano-cylindrical lens and one negative plano-cylindrical lens. The prototype uses a 150mm focal length cylindrical lens and a -19 mm focal length cylindrical lens both manufactured by Melles Griot Inc. These lenses are positioned to form a Galilean telescope. When positioned correctly the cylindrical lenses will not effect the proper operation of the imaging lens. The ratio of the focal length of these lenses is approximately equal to the ratio of the uncorrected field-of-view of the width of the sensor to the desired field-of view. The desired field-of-view, X, is determined based on Equation 2.3, where Y is the separation of the sensor and laser, H is the height of the system above the road, and *Hc* is the desired minimum detectable object height, as shown in Figure 2.11. To insure reliable vehicle detection, it is important that *Hc* be below the bumper height of most common vehicles.

$$Y = \frac{X(H - H_c)}{H_c} \qquad (2\text{-}3)$$

The uncorrected field of view, about 13cm (5 in), results in a critical height of about 1.8 m (6.0 ft). To ensure vehicle detection it is necessary to have a critical height somewhere below the bumper height of the vehicles. A height of around 46cm (18 in) was thought to be acceptable. To achieve this it is necessary to restrict the field of view X to about 2.3cm (0.92 in). This is a factor of reduction of about 5. In our case, where f1 = 150 mm and f2 = -19 mm, the factor of reduction is equal to about -7.9 (the negative sign indicates an inverted image), giving us a field of view of about 16 mm (0.63 in). The factor of reduction is commonly referred to as the angular magnification of the system. As shown in Figure 2.12, a ray of light entering the system from the left at an angle Θ_1 exits the system at the right at an angle Θ_2 equal to Θ_1*(*f1/f2*). Because of this, objects to the left appear to be larger than they actually are. This is how the field of view is reduced. A sensor on the right of the telescopic system will have its field of view reduced by a factor equal to the angular magnification of the system. The telescopic system does not alter the position or focus of the image. Objects that are properly focused by the imaging lens remain in focus when the telescopic system is added.

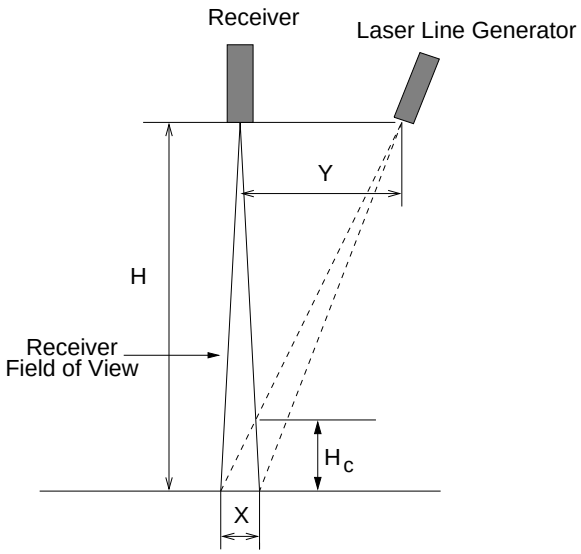


Figure 2.11: Minimal Object Height

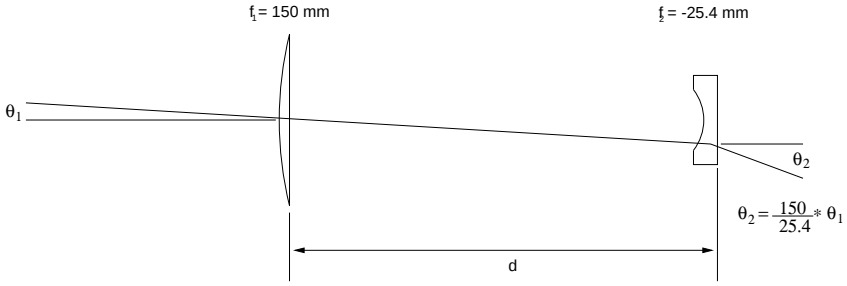


Figure 2.12: Telescope Lens System

The filter is mounted between the telescopic and imaging lens systems. This filter is mounted on a ring that is threaded onto the front of the imaging lens.

2.2 System Electronics

According to the principle of detection, the system needs only to distinguish vehicle presence under the laser lines. It is good to simplify the system by providing a digital output signal from the electronic circuitry. Using a digital signal as output from the hardware will significantly simplify the signal processing in the software, therefore reducing the requirements of the computer system. The implementation of this method is based on the high signal-to-noise ratio of the new circuitry. A block diagram of the new version of the field prototype hardware construct is shown in Figure 2.13. The hardware consists of five parts: the power supply, the clock generator, the laser components, the amplifiers and signal processing circuit, and the digital I/O board and computer.

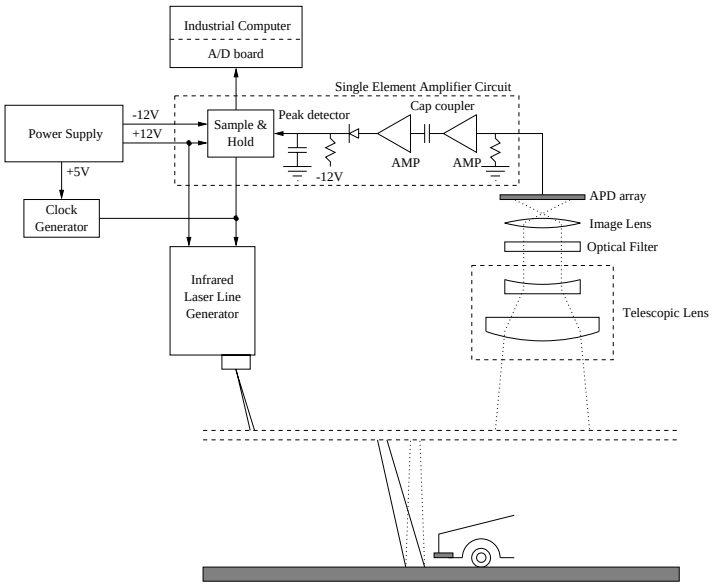


Figure 2.13: Diagram of Electronics Hardware

The power supply section delivers power to both the drivers of the diode lasers and sensor circuits. There are several different voltages needed by the system. A triple output switch-type power supply provides ±12 V, and +5 V. The advantages of using a switch-type power supply are small size, low ripple and high efficiency, compared to a voltage regulator. The +5 V output is used to power the clock generator and digital part of sensor circuit. The +12 V output supplies power to the laser system and the DC/DC converter required by the sensor array. A high voltage DC/DC converter changes 12 V DC to 250-350 V DC, which is used to bias the sensor array from –290V to -310V. The amplifiers use the ± 7.5 V power, which is obtained by two adjustable voltage regulators LM317 and LM318. The input of the voltage regulator is ± 12V from a switch-type power supply.

The high voltage pulse generator for the laser diode is built into the laser unit, which is powered by 12 V DC. Because the pulse generator is isolated by a transformer and is well shielded, there is very low noise induced by the pulse. The power supply for the laser units is well isolated by filters. As a result, even though we used only one power supply for both laser and sensor electronics, there is no interference between the parts. This reduces the cost of the system.

The clock generator provides a clock signal that is used to trigger the laser. An NE555 is used as the oscillator, as shown in Figure 2.14. The frequency can be adjusted from the range of 1Hz to 50kHz. The frequency and width of the pulse can be chosen by adjusting the values of resistors R_a and R_b.

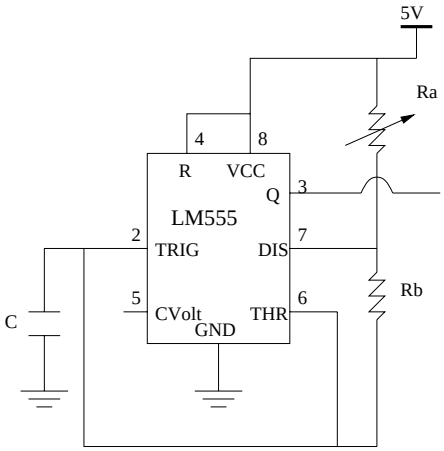


Figure 2.14: Clock Circuit

A 25-element avalanche photodiode (APD) array is used as the sensor in our detection system. The sensor converts the reflected laser light into a current signal. The sensor circuit is the main part of the electronic hardware in the detection system. In this new version of electronic hardware, some low-cost amplifier chips with suitable bandwidth which can meet the high demand of our detection system were chosen for signal amplification. The time response of the new circuit provides a good match to the pulse length of the laser. The high-frequency signal can be amplified effectively without oscillation. The new circuit increases the signal-to-noise ratio by a factor of 5 relative to the previous one. A new method using a TTL logic circuit, instead of a sample-and-hold amplifier, has been used to handle the short signal pulse. Using this method, a TTL logic circuit is triggered by the amplified signal and the output is a digital signal. Usually, the sample-and-hold amplifier is the bottleneck of the time response of the circuitry, so this method will improve the time response reliability of the system and allow us to gather more channels of the signal, for example up to 24 channels. Using a digital signal as output from the hardware will significantly simplify the signal processing in the software. The implementation of this method is based on the high signal-to-noise ratio of the new circuitry. The new circuitry is based on simple, cheap, commonly available electronic components, so the cost of new circuitry is only one-fifth of that of the previous one. This is meaningful for the commercialization of our system in the future.

The circuit can be divided into three stages: signal amplification, interface and signal conditioning, and digital output, as shown in Figure 2.15.

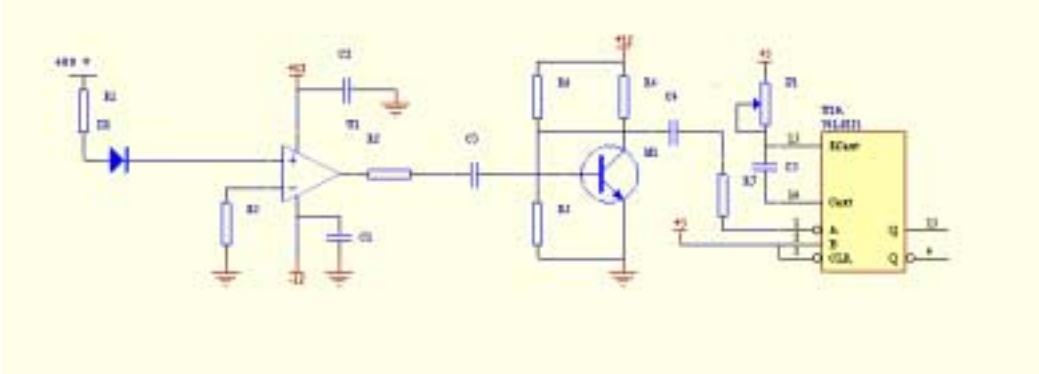


Figure 2.15: Circuitry of Sensor Electronics

The current generated by laser light reflected onto the sensor element is amplified by a video amplifier U1. The output of this amplifier is about 1 volt. A high-speed transistor T1 is used for signal conditioning and provides an interface between analog and digital parts of the circuit. A Multi-Vibration Mono-Stable Oscillator U2 is triggered by the pulses from amplifier and generates high level output when there is no vehicle under the system. When the moving vehicle blocks the laser, there is no pulse to trigger the U2, and the output will be low. Capacitors are used to isolate DC signals between different parts of circuit, so that a continuous signal can not pass through the circuit. Normally, ambient lights generate continuous signals. After intensive tests and adjustment, the interference between different part of circuits are eliminated to the lowest level, so that the system has a signal/noise ratio 5 times higher than the previous version. `Figure 2.16` shows a typical signal from the amplifiers.The transition time is an important factor to the accuracy and consistency of data measured. Because the new version of electronic circuitry generates signals directly in the digital mode, the transition time, which contributed to the error in the measurement in the previous version of the prototype, is no longer a problem in the new system.

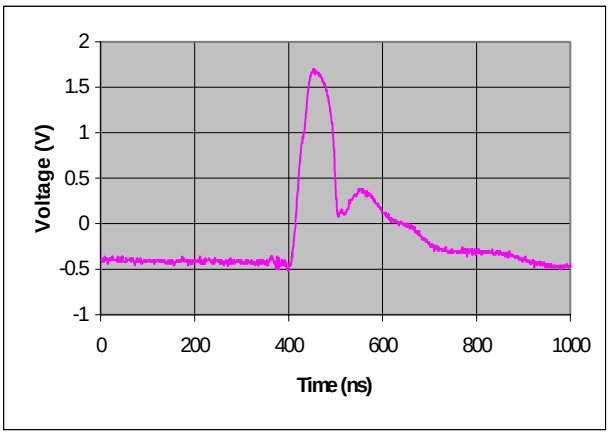


Figure 2.16: Output of Amplifier

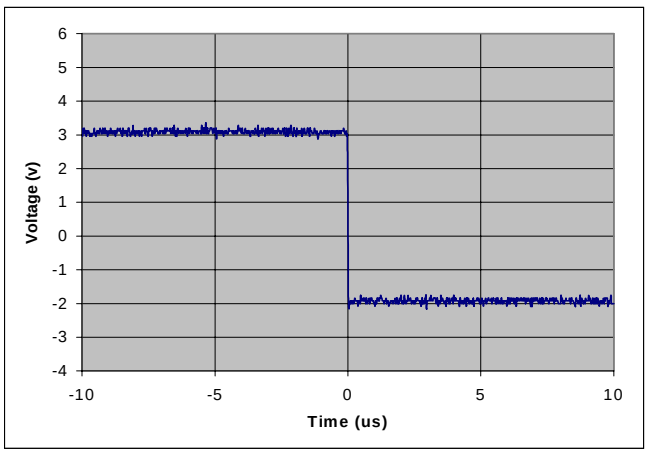


Figure 2.17: Transition of Digital Output

Because of the use of surface mounted chips and simplicity of the circuitry, the new version electronics has a compact size. This makes it possible to place all amplifiers for 24 channels on a small printed circuit board without long connection wires. This is important because the signal from the photodiode array is very weak. The longer the connection wires between the photodiode array and amplifiers, the more noise and oscillation will be introduced into the signal. Figure 2.18 and Figure 2.19 are the printed circuit boards for amplifiers and sensor mount. The circuit board layouts are optimized to reduce cross talk between channels.

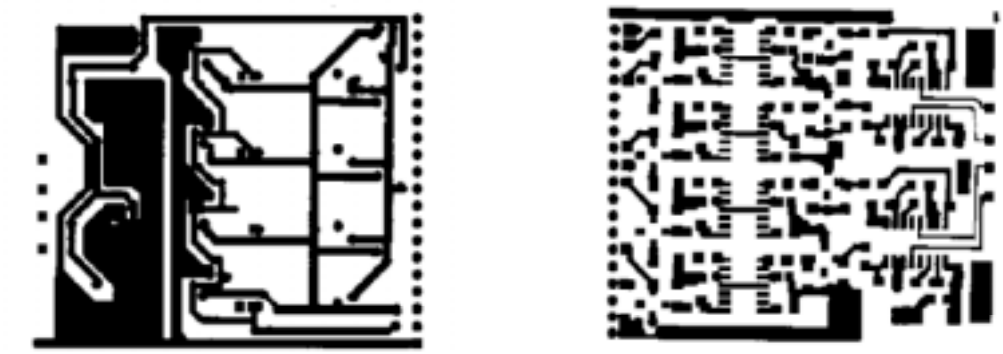


Figure 2.18: Printed Circuit Boards for High-Speed Amplifiers

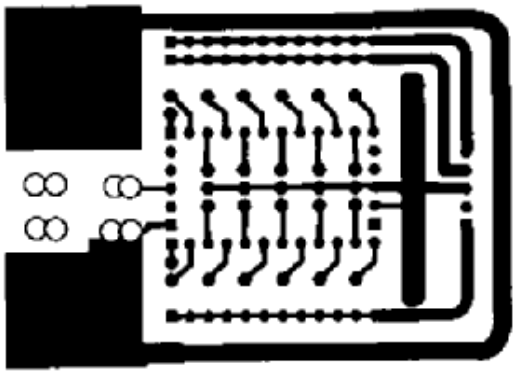


Figure 2.19: Printed Circuit Board for Sensor Mount

Figure 2.20 illustrates the modularized printed circuit board mount. This structure allows us to use all 24 elements of the sensor array and is easy to maintain.

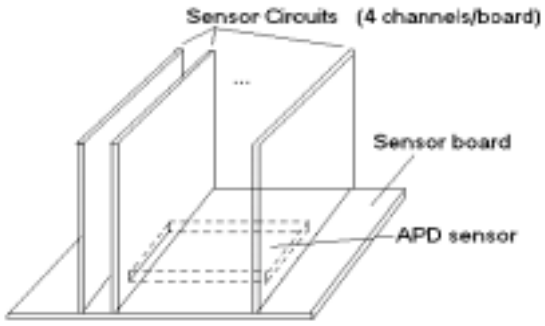


Figure 2.20: Mount of Print Circuit Boards

2.2.1 Data Acquisition and Computer System

Since the outputs of the sensor circuits are digital signals, the A/D converter is no longer needed. A general-purpose low-cost digital I/O board PCI-DIO-96 from National Instruments is used an input interface between the sensor circuitry and the computer system. This digital I/O has 96 channels of configurable input/output, so it is suitable for our system when all 48 channels of signal are required. Normally digital I/O is fast enough to transfer data on the order of kHz.

2.3 Mechanical Design

Even though all the optical components used in the new version of the system are the same as the previous system, the mechanical structure holding the optical components is completely redesigned. The new mechanical design of optics is flexible and allows the optical components to be adjusted to optimize alignment and focus of the laser beam. The mechanical configuration of system is shown in Figure 2.21. The cylindrical lenses, image lenses and APD sensors are mounted on optical rails, which are placed in the base plate. Each optical component and sensor is clamped by a holder that is adjustable in two dimensions. The image lenses can be swung around an axis with an optical cell. The sensor is clamped by a ring which is mounted on a rotation lens holder. As a result the sensor can be adjusted via both translation and rotation.

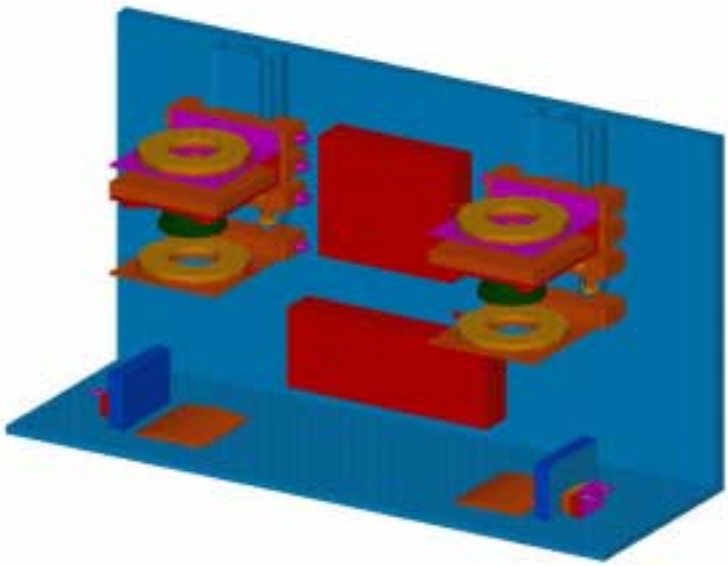


Figure 2.21: Mechanical Configuration of the Detection System

Because the new optical system is flexible, it is necessary to have the procedures to adjust the system. In the optical system shown in Figure 2.22, the adjustment includes optimizing the cylindrical telescope, focusing, lining up optical components and lasers. We have developed procedures for optical adjustment of the system as following:

- Line-up telescope lenses.
- Adjust the vertical alignment of the two sensor arrays.
- Adjust the magnification of the optical system.
- Align the laser to the optic's centerline.
- Find the image focusing distance.

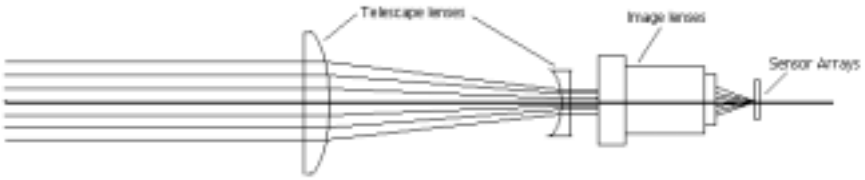


Figure 2.22: Optical System

After adjustment following the defined procedures, a clear image of the reflecting laser can be obtained. The optimization of the optics significantly improves the critical height and eliminates the unexpected reflections when some reflective vehicle surfaces are present under the detector. A visible laser and a CCD are used to verify the image quality of the optical system. Figure 2.23 shows the image of a reflected laser line on the position of a sensor from different heights. The principle of critical height is clearly demonstrated in this figure.

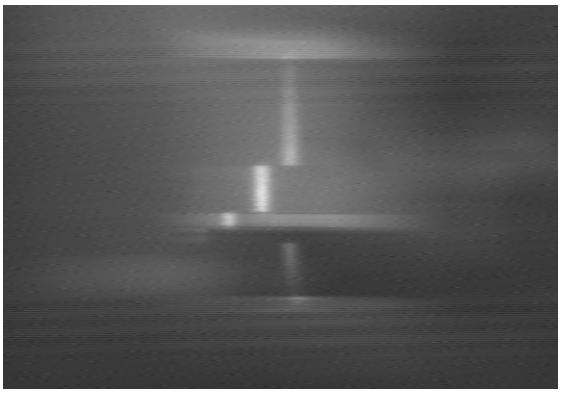


Figure 2.23: Image of Laser on the Sensor

Chapter 3 Real-Time Data Acquisition and Processing Software

The data acquisition software is used to collect, process and display data from hardware of the detection system. The data has to be read from the output of sensor electronic circuit periodically. All tasks have to be done at the same time. However the data has to be acquired in a same interval from the hardware. In other words the data acquisition task should be deterministic, otherwise the information will loss. The concurrence of the deterministic tasks requires real-time performance of operating system. Previously we were using a real-time operating system Lynx OS. Because of its incompatibility with other systems, lack of third party support and high-cost, we have developed new software based on the Microsoft Windows NT system with real-time extension form RTX from Venture.com, Inc. The new software utilizes the hard real-time character of the RTX subsystem, while taking advantage of the convenience of API from Windows NT system for non-real-time processing of data.

The architecture of the RXT real-time extension is shown in Figure 3.1. RTX adds a real-time subsystem, known as RTSS, to Windows NT. RTSS is conceptually similar to other Windows NT subsystems (such as Win32 and DOS). Instead of using the Windows NT scheduler, RTSS performs its own real-time thread scheduling. Furthermore, all RTSS thread scheduling occurs ahead of all Windows NT scheduling. By this way RTX delivers a hard real-time on the NT system. In our detection system, data from the sensors are collected using real-time process running in the RTX environment, and are then processed in a non-real-time Win32 process.

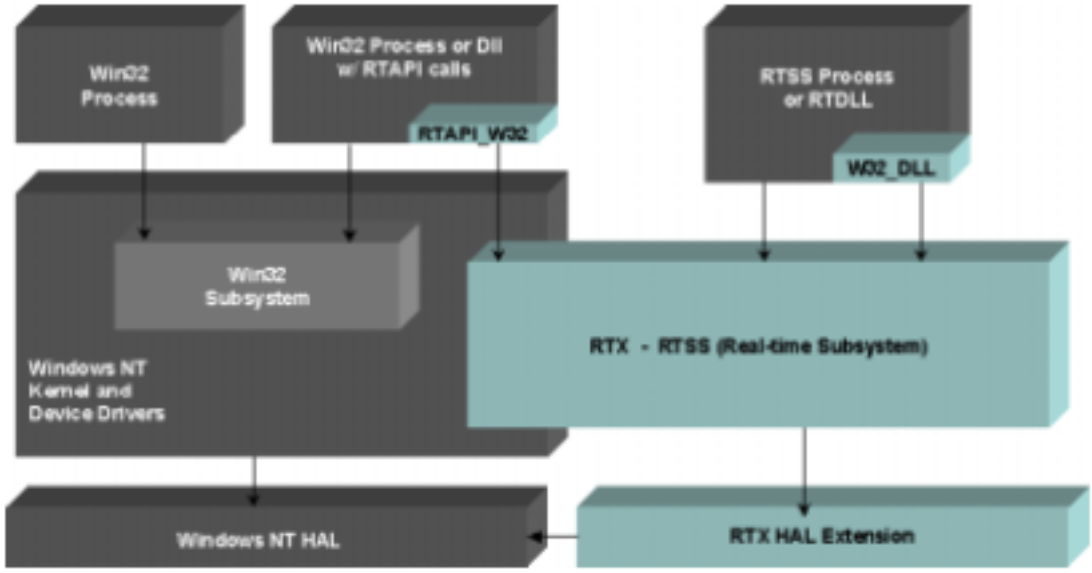
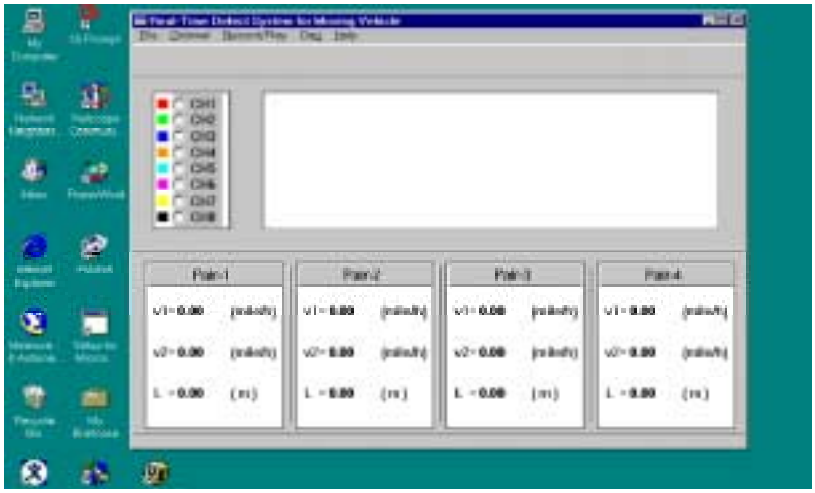


Figure 3.1: System Architecture of RTX Real-Time Extension

RTX provides a set of APIi for accessing hardware from RTX real-time process. This makes development of device drivers easier. The device driver has been developed for the PCI-DIO-96 digital I/O board in the real-time RTX subsystem to acquire the data at a rate up 10 kHz with highest priority. The driver provides a shared memory for inter-process communication and semaphore for synchronization with Win32 non-real-time process. The double-buffering technique is used to collect data in the shared memory so that no data will be lost.

The data processing includes calculation of speeds and lengths of vehicles, displaying graphic data on the screen and storing data to the disk. Figure 3.1 is the graphical user interface running on Win32 processing. The application program accesses real-time RTX processes though shared memory and semaphore. The program can acquire data from the I/O board, calculate lengths and speeds of vehicles, and store to data to disk (for later replay), exactly the same as the last version software based on the Lynx OS.



Equation 3-1: User Interface on Windows NT

Software simulation of signals from detection system has been developed for testing of the data acquisition and processing software. The simulation software has been written in the RTX real-time subsystem and the simulation signals can be output from the parallel port of the PC. It accurately simulates 8 channels of signals and can be extended to 48 channels by increasing the count number of the loop. This simulation software allows a developer to test and debug data acquisition software separately from the hardware of the detection system.

Chapter 4 Field Test Results

Field tests have been made on the highway with real traffic near the junction of highway I-5 and I-80 in Sacramento. Figure 4.1 is a picture of the test site and the detection system mounted on the bridge across the highway.

Figure 4.1: Detection System Mounted Above the Highway

In the current phase, only four of twenty-four elements of each sensor were used for testing. The front and rear speeds, length and acceleration were obtained in the tests according to the measured data. In the last highway test, the older data acquisition system was still used (in conjunction with the new detector system). The sampling rate of data acquisition was set to 1.8kHz to avoid loss of data. The frequency of the laser was 7kHz. The results indicated that the signals of the new version of the system are clear and the transition is fast enough in the system. The maximum error of the measurement is less than 10%, which is caused by the low sampling rate. The error will decrease at higher sampling rate. The asynchronicity between sampling in the computer and clock of laser pulse also partly contributed to the errors. Figure 4.2 demonstrates the typical signals and measurements of highway test. Because the distance between the two lasers was not calibrated accurately, the speed displayed in the figure is higher than the actual speed. Comparing with the results from previous version of detection system, as shown in Figure 4.3, the signals from new system are much clearer and faster than the old one. This makes it possible to obtain the measurement accuracy required by this project.

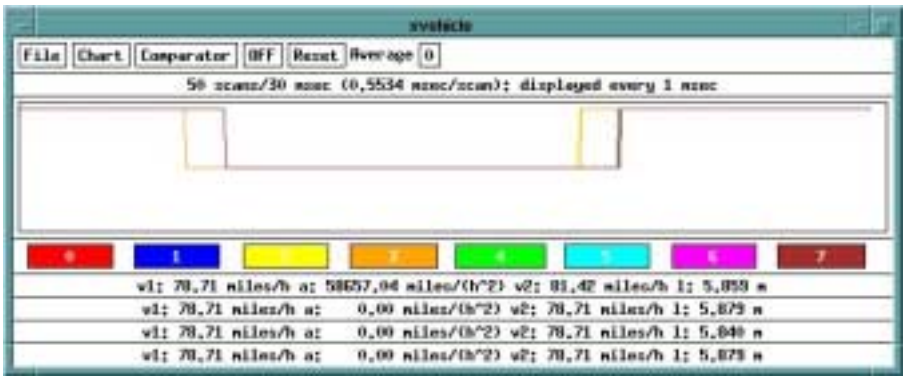


Figure 4.2: Test Results

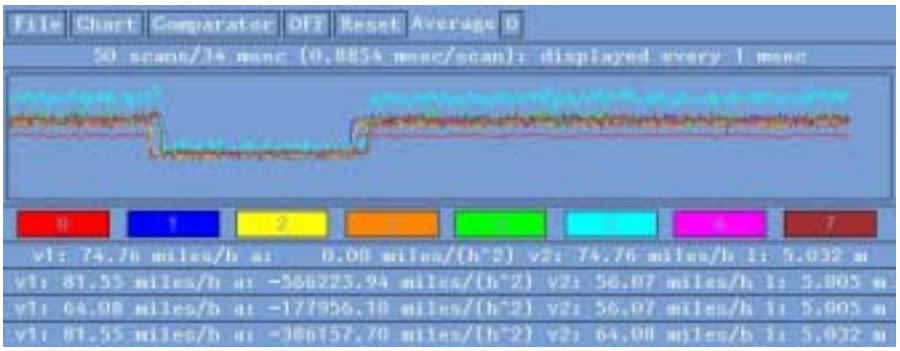


Figure 4.3: Test Results from Previous Version of Detection System

Chapter 5 Future Work

We will continue to improve the field prototype system according to the problems that surfaced in the highway tests. For example, we will modify the current mechanical design so that it will be rigid enough to keep the optical alignment unchanged when the system is moved from the lab to the highway. The new version of the system will be able to gather more channels of the signal, for example up to 24 channels, which is necessary to obtain precise profiles of moving vehicles and to deal with the boundary effects that occur when the vehicles are changing lanes.

The electronics will be tested in wide ranges of temperature and humidity. The Signal/Noise ratio changes in different road materials and conditions will be obtained for further improvement of electronic circuitry including gain control and circuit board layout. We will assess different ways of assuring weather resistance while allowing component accessibility for maintenance, such as box designs, gasket enclosures, and nitrogen pressurization.

Future study will also be aimed at reducing the cost, building a robust system, and dealing with some extreme road and weather conditions. For the software we will develop new software and algorithms to handle different situations of moving vehicles and calibration of reflectance signatures of moving vehicles. Various scenarios of motion will be studied. For example, the laser beam may be adjusted so that it is not exactly orthogonal to the vehicle velocity vector.

We are developing software for the new version of the system using a free Linux operating system. RTlinux will be used as a real-time kernel, as it is becoming more popular in real-time embedded applications. RTlinux allows deterministic real-time and non-real-time processing within the same computer. The device driver for the data acquisition board will be developed in RTlinux and the data processing and display program will be ported from LYNX OS to the Linux system.

Chapter 6 Conclusions

We have developed and tested a new version of field prototype of a laser-based real-time, non-intrusive detection system for measurement of delineations of moving vehicles in real traffic environments. This field prototype has substantial improvement in both electronics and optics over the previous system. The signal level and transition time are no longer a problem in the new system. This enables us to concentrate on the higher level objectives of the project, such as dealing with different traffic scenery and some kinds of special vehicles. The test results quantitatively verified that the principle of our detection system is technically sound and indicated that the algorithm implemented in the software works in most cases. The simple method of detecting vehicle presence based on the absence of reflected laser beam works reliably in the real traffic environment. The real-time data acquisition software has been developed to gather and process the data from system hardware based on Windows NT operation system with RTX real-time extension. The vehicle delineation library was used to convert data from the sensor library, which was developed in previous project, to vehicle delineation data, such as front and real velocities of vehicle, average acceleration and length. The speed, acceleration, and length of a detected vehicle can be displayed on the screen simultaneously. All these data of the detected vehicle can be saved with a time stamp to a disk in real time. The new software possesses the same functionality while having much more simplicity.

Bibliography

[1] Palen, J., *Roadway Laser Detector Prototype Design Considerations, personal communications,* 1996.

[2] Tyburski, R.M., *A Review of Road Sensor Technology for Monitoring Vehicle Traffic*, ITE journal. Vol. 59, no. 8 (Aug. 1989)

[3] Halvorson, G. A., *Automated Real-Time Dimension Measurement of Moving Vehicles Using Infrared Laser Rangefinders*, MS Thesis, University of Victoria, 1995.

[4] Olson, et al., *Active Near-Field Object Sensor and Method Employing Object Classification Techniques*, U.S. Patent No. 5,321,4990, 1994.

[5] Wangler, et al., *Intelligent Vehicle Highway System Sensor and Method*, U.S. Patent No. 5,546,188, 1996.

[6] Wangler, et al., *Intelligent Vehicle Highway System Sensor and Method, U.S. Patent* No. 5,757,472, 1998.

[7] Wangler, et al., *Intelligent Vehicle Highway System Multi-Lane Sensor and Method*, U.S. Patent No. 5,793,491, 1998.

[8] Graeme, J., *Photodiode Amplifiers: Op Amp Solutions*, McGraw-Hill, New York, 1996.

[9] Horowitz, P. & Hill, W., *The Art of Electronics*, 2nd Ed., Cambridge University Press, New York, 1989.

[10] *American National Standard for the Safe Use of Lasers*, Laser Institute of America, Orlando, 1986.

[11] Kirk Van Katwyk, *Design and Implementation of Real-time Software for Open Architecture Integration of Mechantronic System*, thesis for Master of Science, Department of Mechanical and Aeronautical Engineering, University of California at Davis, 2000.

[12] Jonathan E. Larson E. Kirk Van Katwyk, Cheng Liu, Harry Cheng, Ben Shaw, Joe Palen, *A Real-Time Laser-Based Prototype Detection System for Measurement of Delineation of Moving Vehicles*, California PATH Working Paper, UCB-ITS-PWP-98-20, September 1998.

Chapter 7 Appendix

7.1 Device Driver for Digital I/O on Windows NT

```
/*
 * Nodule Name:
 *
 *       DIO96_gather.c
 *
 * Abstract:
 *
 * Gather data from DIO-96 board in a RTX process
 *
 * Author:
 *
 *       Bin Lin, 20-April-2000
 *
 * Environment:
 *
 *       RTX real-time extension.
 *
 * Revision History:
 *
 */

#include "windows.h"
#include "stdio.h"
#include "rtapi.h"
#include "DIO96.h"

#define NO_ERRORS  0
#define ERROR_OCCURED -1
#define PTL(x) (*(PLONG)(x))

PCHAR _base_addr;
PCI_SLOT_NUMBER     SlotNumber;
PPCI_COMMON_CONFIG  PciData;
UCHAR               buffer[PCI_COMMON_HDR_LENGTH];

int index = 0;

//for IPC
static HANDLE    hShm, hSemPost, hSemAck;
static BOOL      bInit = FALSE;
static PMSGSTR   pMsg;
LONG     abandoned = 0;
int              finished = 0;
LONG     lReleaseCount = 1;
UCHAR Data;

int RTFCNDCL TimerHandler (PVOID unused);

PCHAR SetupDIO96();   //Setup DIO-96 board

  PCHAR vBAR0;

int
main( int argc, char *argv[] )
{
  LARGE_INTEGER    Period;        // Timer period
  HANDLE          hTimer;
  Period.QuadPart = 100;

  //Setup DIO96
  _base_addr = SetupDIO96();
```

```
printf("_base_addr = 0x%p\n", _base_addr);

//set port A, B and C as input
*CNFG_ADDR = 0x80;

printf("CNFG_ADDR = 0x%x\n", *CNFG_ADDR);

  // Open the required IPC objects upon first call.
  if (!bInit)
  {
        hShm = RtOpenSharedMemory( PAGE_READWRITE, FALSE, MSGSTR_SHM, (LPVOID) &pMsg);
        if (hShm==NULL)
        {
               printf("DIO96_gether.c: Error:  Could not open Shared Memory.  GetLastError =
                      %d\n", GetLastError());
               return FALSE;
        }

        hSemPost = RtOpenSemaphore( SYNCHRONIZE, FALSE, MSGSTR_SEM_POST);
        if (hSemPost==NULL)
        {
               printf("DIO96_gather: Error:  Could not open Semaphore.  GetLastError = %d\n",
                      GetLastError());
               RtCloseHandle(hShm);
               return FALSE;
        }
}

        hSemAck = RtOpenSemaphore( SYNCHRONIZE, FALSE, MSGSTR_SEM_ACK);
        if (hSemAck==NULL)
        {
               printf("MsgClient: Error:  Could not open Semaphore.  GetLastError = %d\n",
                       GetLastError());
               RtCloseHandle(hShm);
               RtCloseHandle(hMutex);
               RtCloseHandle(hSemPost);
               return FALSE;
        }

        bInit = TRUE;
  }


//Setup and start a periodic timer.

if (!RtSetThreadPriority(GetCurrentThread(), RT_PRIORITY_MAX-1))
       printf("WARNNING: Can't set to highest RTAPI priority. \n");

if (!(hTimer = RtCreateTimer(NULL,    // Security - NULL is none
                             0,     // Stack size - 0 is use default
                             TimerHandler,      // Timer handler
                             NULL,  // NULL context (argument to handler)
                             RT_PRIORITY_MAX,  // Priority
                             CLOCK_2)))  // Always use fastest available clock
  {
      printf("Srtm: Error: Could not create the timer.  GetLastError = %d\n",
            GetLastError());
       return ERROR_OCCURED;
  }

  if (!RtSetTimerRelative( hTimer, &Period, &Period))
  {
      printf("SRTM: ERROR: Could not set and start the timer.  GetLastError = %d\n",
             GetLastError());
       RtDeleteTimer( hTimer);
       return ERROR_OCCURED;
  }


*PORTA_ADDR=0x0;
```

```
  // Wait for the sampling time and then stop the timer.
  Sleep(5000);

        if(!RtDeleteTimer( hTimer))
        {
                printf("Srtm: Error: Could not delete timer.  GetLastError = %d\n",
                        GetLastError());
                return ERROR_OCCURED;
        }

  printf("PORTA_ADDR = %p\n", PORTA_ADDR);
  printf("PORTA = 0x%x\n", *PORTA_ADDR);
  printf("vBAR0 = 0x%x\n", *vBAR0);
  printf("vBAR0+3 = 0x%x\n", *(vBAR0+0x03));
  printf("Buffer[0] = 0x%02x\n", *(pMsg->Buffer));
  return NO_ERRORS;
}


//
// timer handler function
//
int RTFCNDCL TimerHandler (PVOID   unused)
{
  // Test sampling rate
  *PORTC_ADDR = 0x00;

  pMsg->Buffer[index] = *PORTA_ADDR;
  index++;

  if(index == BUFFER_SIZE/2)
  {
        pMsg->BufFull = 0;
        if(!RtReleaseSemaphore(hSemPost, lReleaseCount, NULL))
        {
        printf("DIO96_gather: Error: Could not release semaphore.  GetLastError = %d\n",
                GetLastError());
        return FALSE;
        }
  }

  if(index == BUFFER_SIZE)
  {
        pMsg->BufFull = 1;
        if(!RtReleaseSemaphore(hSemPost, lReleaseCount, NULL))
        {
        printf("DIO96_gather: Error: Could not release semaphore.  GetLastError = %d\n",
                GetLastError());
        return FALSE;
        }
  index=0;
  }

  *PORTC_ADDR = 0xff;
  return NO_ERRORS;
}


PCHAR SetupDIO96()
{
    ULONG       i;                  // logical slot number for the PCI adapter
    ULONG       f;                  // function number on the specified adapter
    ULONG       bytesWritten;       // return value from RtGetBusDataByOffset
    ULONG       bus;                // bus number
    BOOLEAN     flag;
  ULONG          Offset= 0;
  ULONG          nothingWritten = 0;
  ULONG          AddressSpace = 0;
  ULONG          window_data_value;
  //PCHAR vBAR0;
  PCHAR vBAR1;
```

```
LARGE_INTEGER BAR0;
LARGE_INTEGER BAR1;
LARGE_INTEGER tBAR0;
LARGE_INTEGER tBAR1;

BAR0.QuadPart = 0;
BAR1.QuadPart = 0;
tBAR0.QuadPart = 0;
tBAR1.QuadPart = 0;

PciData = (PPCI_COMMON_CONFIG) buffer;
  SlotNumber.u.bits.Reserved = 0;
  flag = TRUE;

  for (bus=0; flag; bus++) {

      for (i=0; i<PCI_MAX_DEVICES && flag; i++) {
          SlotNumber.u.bits.DeviceNumber = i;

          for (f=0; f<PCI_MAX_FUNCTION; f++) {
              SlotNumber.u.bits.FunctionNumber = f;

              bytesWritten = RtGetBusDataByOffset (
                  PCIConfiguration,
                  bus,
                  SlotNumber.u.AsULONG,
                  PciData,
                  Offset,
                  PCI_COMMON_HDR_LENGTH
                  );

              if (bytesWritten == nothingWritten) {
                  // out of PCI buses
                  flag = FALSE;
                  break;
              }

              if (PciData->VendorID == PCI_INVALID_VENDORID) {
                  // no device at this slot number, skip to next slot
                  break;
                       }
                      if((PciData->VendorID == 0x1093) &&(PciData->DeviceID == 0x0160))
                      {
                          BAR0.QuadPart = PciData->u.type0.BaseAddresses[0];
                          BAR1.QuadPart = PciData->u.type0.BaseAddresses[1];

                          //Traslate the base port address
                          if(!RtTranslateBusAddress(PCIBus,0,BAR0,&AddressSpace,&tBAR0))
                          {
                             printf("tBAR0 translation failed\n");
                             }
                          else
                          {
                             printf("tBAR0:SystemMappedAddress: 0x%08x\n", tBAR0.LowPart);
                             printf("BAR1.QuadPart = 0x%08x\n",BAR1.QuadPart);
                          }
                          if(!RtTranslateBusAddress(PCIBus, 0,BAR1,&AddressSpace,&tBAR1))
                          {
                             printf("tBAR1 translation failed\n");
                          }
                          else
                          {
                             printf("tBAR1:SystemMappedAddress: 0x%08x\n", tBAR1.LowPart);

                          }

                          //Map the address to virtual address the software can use

                          vBAR0 = RtMapMemory(tBAR0, 4*1024,0);
                          if (vBAR0==0)
```

```
                                {
                                   printf("vBAR0:Failure on RtmapMemory\nError=n", GetLastError());
                                }
                                else
                                {
                                   printf("vBAR0: Virtual Memory Address: 0x%08x\n", vBAR0);
                                }
                                vBAR1 = RtMapMemory(tBAR1, 4*1024,0);
                                printf("vBAR1 = 0x%x\n", vBAR1);
                                if (vBAR0==0)
                                {
                                   printf("vBAR1:Failure on RtmapMemory\nError=n", GetLastError());
                                }
                                else
                                {
                                   printf("vBAR1: Virtual Memory Address: 0x%08x\n", vBAR1);
                                }

                                //Setup
                                PciData->Command = (PCI_ENABLE_IO_SPACE|
                                                    PCI_ENABLE_MEMORY_SPACE|
                                                    PCI_ENABLE_BUS_MASTER|
                                                    PCI_ENABLE_WRITE_AND_INVALIDATE);

                                RtSetBusDataByOffset(PCIConfiguration, bus,
                                                    SlotNumber.u.AsULONG, PciData,0,
                                                    PCI_COMMON_HDR_LENGTH);

                                window_data_value =((0xffffff00&(ULONG)BAR1.LowPart)(0x00000080));
                                PTL(vBAR0+0x000000c0) = window_data_value;

  // A device is found, print out the PCI configuration information
                                printf("\nPciData: ------------------------\n");
                                printf("VendorID:\t\t0x%x\n", PciData->VendorID);
                                printf("DeviceID:\t\t0x%x\n", PciData->DeviceID);
                           } //dio96
                    }
                }
              }
  return vBAR1;
}
```

7.2 Application Program on Windows NT

```
/*********************************************************************
 * main.cpp
 *
 * Laser-Detector System: Main Program
 *
 *********************************************************************/
#include "stdafx.h"
#include "resource.h"
#include "rtapi.h"
#include "daq.h"
#include "daqlib.h"

#define  DX 8          //Displacement of scrollwindow
#define T0           (1L)            //receive the first edge
#define T1           (1L)<<1  //receive the second edge
#define T2           (1L)<<2  //receive the third edge
#define T3           (1L)<<3  //receive the fourth edge
#define N_DIODE      8               //number of photodiode
#define N_PAIR       4               //pair of photodiode
#define DISTANCE     254/(1609*1e3)          //distance of two array (mile)
#define  APP_NAME        "RTX Detector System"
#define  ERROR_CODE      1

#ifdef DEBUG
#define debug(a) printf a
#else
#define debug(a)
```

```
#endif

char    szAppName[] = "menudemo" ;
struct                          //the type and name of child window
     {
       long style ;
       char *text ;
     }
     button[] =
     {
        BS_AUTOCHECKBOX,    "CH1",
        BS_AUTOCHECKBOX,    "CH2",
        BS_AUTOCHECKBOX,    "CH3",
        BS_AUTOCHECKBOX,    "CH4",
        BS_AUTOCHECKBOX,    "CH5",
        BS_AUTOCHECKBOX,    "CH6",
        BS_AUTOCHECKBOX,    "CH7",
        BS_AUTOCHECKBOX,    "CH8",
     } ;
HWND    hwnd ;          //the handle of window
HDC             hdc;            //the handle of device context
static HRGN     hrgnu,          //handle of curve region
                hrgnd,          //handle of pair-1 -- pair-4 data region
                hrgnd1,         //handle of pair-1 data region
                hrgnd2,         //handle of pair-2 data region
                hrgnd3,         //handle of pair-3 data region
                hrgnd4,         //handle of pair-4 data region
                hrgnd5,         //handle of pair-1 data region + pair-2 data region
                hrgnd6;         //handle of pair-3 data region + pair-4 data region
HANDLE   ThreadU, ThreadD;              //handle of thread
HMENU   hMenu ;                                 //handle of menu
HWND    hDlgModeless;
RECT    RePaint;                //scrollwindow rect
PAINTSTRUCT     ps;

BOOL    Pen1=FALSE,             //flag of pen1
        Pen2=FALSE,             //flag of pen2
        Pen3=FALSE,             //flag of pen3
        Pen4=FALSE,             //flag of pen4
        Pen5=FALSE,             //flag of pen5
        Pen6=FALSE,             //flag of pen6
        Pen7=FALSE,             //flag of pen7
        Pen8=FALSE;             //flag of pen8
BOOL    bkill=FALSE;   //flag of thread
BOOL    exmove=FALSE;  //flag of extra move
BOOL    DAQ=FALSE;              //flag of data acquisition
BOOL    NoPlay=FALSE;  //File dosen't exsist

int     Lastpoint=0;   //last drawing point
int             qwrite=0,               //input location
        qread=0,                //output location
        qsize=1280;             //queue size

static int      point1[BUFFER_SIZE/2], //channel-1 array for paint
                point2[BUFFER_SIZE/2], //channel-2 array for paint
                point3[BUFFER_SIZE/2], //channel-3 array for paint
                point4[BUFFER_SIZE/2], //channel-4 array for paint
                point5[BUFFER_SIZE/2], //channel-5 array for paint
                point6[BUFFER_SIZE/2], //channel-6 array for paint
                point7[BUFFER_SIZE/2], //channel-7 array for paint
                point8[BUFFER_SIZE/2]; //channel-8 array for paint
static int      pointLast[8];

double  v[N_PAIR][2],  //front and back velocity of vehicle
        a[N_PAIR],              //acceleration
        l[N_PAIR];              //length of vehicle
//char old = 0xFF;                                      //the last data
//unsigned long edge_status[N_PAIR];  //status of edge, xxxxT3T2T1T0
//int e[N_PAIR];                                                //the order of edge, 0,1,2,3
//unsigned long t[N_PAIR][4];         //edge timing information
enum{                                                   //vehicle direction
```

```
               None,
               Presence,
               Trigger
} edge[N_PAIR];
//double delta_t,              //dt
//       delta_v;              //dv
int iFile=0;
static char szFile[256]="", szFileTitle[256]="";

GLOBALHANDLE        /*hpt1,hpt2,hpt3,hpt4,
                              hpt5,hpt6,hpt7,hpt8,*/
                              hqueue = NULL;
//int   *pt1,*pt2,*pt3,*pt4,*pt5,*pt6,*pt7,*pt8;
unsigned char   **dpqueue = NULL;
static unsigned char onebuf[BUFFER_SIZE/2];

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL    CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;
void MAlloc();
void MFree();
void Plot();
void MsgAndExit(PCHAR string);
void DaqStart();
void DaqEnd();
void Play();
void Record();
void RPStop();
void GetFileName();

int APIENTRY WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      PSTR szCmdLine, int iCmdShow)
{
     MSG        msg ;
     WNDCLASSEX wndclass ;

     wndclass.cbSize        = sizeof (wndclass) ;
     wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc   = WndProc ;
     wndclass.cbClsExtra    = 0 ;
     wndclass.cbWndExtra    = 0 ;
     wndclass.hInstance     = hInstance ;
     wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName  = MAKEINTRESOURCE(IDR_MENU1);
     wndclass.lpszClassName = szAppName ;
     wndclass.hIconSm       = LoadIcon (NULL, IDI_APPLICATION) ;

     RegisterClassEx (&wndclass) ;

     hwnd = CreateWindow (szAppName, "Real-Time Detect System for Moving Vehicle",
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL, hInstance, NULL) ;

     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;

  s=sensorOpen( );

  while (GetMessage (&msg, NULL, 0, 0))
 {
        if(hDlgModeless==NULL || !IsDialogMessage (hDlgModeless,&msg))
         {
           TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
         }
    }
     return msg.wParam ;
}
```

```
void MsgAndExit(PCHAR string)
{
    MessageBox( NULL, string,  APP_NAME, MB_OK | MB_ICONERROR | MB_SYSTEMMODAL);
    ExitProcess(ERROR_CODE);
}

unsigned long WINAPI Thread96U (PVOID pvoid)
{
 DWORD   dwMaximumSizeHigh = 0;
 LONG    lInitialCount = 0;
 LONG    lMaximumCount = 1;
 DWORD retcode=0;
 int i=0, j=0;

 SetThreadPriority(GetCurrentThread(), PRIO_DAQU);

 hShm = RtCreateSharedMemory( PAGE_READWRITE, dwMaximumSizeHigh, sizeof(DAQSTR), DAQ_SHM, (void
**) &pDaq);

 if(GetLastError()==ERROR_ALREADY_EXISTS)
        MsgAndExit("Warning!\nThe shared memory does already exist.");

    if (hShm==NULL)
        MsgAndExit("RtCreateSharedMemory failed.");

    hSemPost = RtCreateSemaphore( NULL, lInitialCount, lMaximumCount, DAQ_SEM_POST);

    if (hSemPost==NULL)
         MsgAndExit("RtCreateSemaphore for posting failed.");

 qwrite=0;
 qread=0;

 for(i=0; i<BUFFER_SIZE/2; i++)
 {
        point1[i] = 100;
        point2[i] = 100;
        point3[i] = 100;
        point4[i] = 100;
        point5[i] = 100;
        point6[i] = 100;
        point7[i] = 100;
        point8[i] = 100;
 }

 /*for (i=0;i<qsize;i++)
 {
        for (j=0;j<100;j++) dpqueue[i][j] = 0xFF;
        for (j=100;j<102;j++) dpqueue[i][j] = 0xFE;
        for (j=102;j<500;j++) dpqueue[i][j] = 0xEE;
        for (j=500;j<501;j++) dpqueue[i][j] = 0xEF;
        for (j=501;j<513;j++) dpqueue[i][j] = 0xFF;
 }*/

while(!bkill)
{
 if(RtWaitForSingleObject( hSemPost, INFINITE) == WAIT_FAILED)
                MsgAndExit("RtWaitForSingleObject failed.");
 if(pDaq->HalfBufFull == 1)
        for(i=0;i<BUFFER_SIZE/2;i++)
                dpqueue[qwrite][i]=pDaq->Buffer[i];
 if(pDaq->HalfBufFull == 2)
        for(i=0;i<BUFFER_SIZE/2;i++)
                dpqueue[qwrite][i]=pDaq->Buffer[BUFFER_SIZE/2+i];
 qwrite++;
 qwrite%=qsize;

 if (s->status & RECORD)
 {
 if(pDaq->HalfBufFull == 1)
```

```
    fwrite(pDaq->Buffer,sizeof(unsigned char),BUFFER_SIZE/2,s->file);
 if(pDaq->HalfBufFull == 2)
        fwrite(pDaq->Buffer+BUFFER_SIZE/2,sizeof(unsigned char),BUFFER_SIZE/2,s->file);
 if(ferror(s->file))
   perror("Record: fwrite()"), exit(1);
 fflush(s->file);
 }

}
 //InvalidateRgn (hwnd, hrgnu, TRUE) ;
 ResumeThread(ThreadD);
 RtCloseHandle(hShm);
 RtCloseHandle(hSemPost);
 ExitThread(retcode);
 //CloseHandle(ThreadU);
 MFree();
 return 0;
}

unsigned long WINAPI Thread96D(PVOID pvoid)
{
 LARGE_INTEGER    Period;        // Timer period
    HANDLE          hTimer;
 static BOOL    bInit = FALSE;
 DWORD retcode=0;

 Period.QuadPart = 100;
 _base_addr = SetupDIO96();
 //printf("_base_addr = 0x%p\n", _base_addr);

 //set port A, B and C as input
 *CNFG_ADDR = 0x92;
 //printf("CNFG_ADDR = 0x%x\n", *CNFG_ADDR);

 if (!RtSetThreadPriority(GetCurrentThread(), PRIO_DAQD))
        MsgAndExit("WARNNING!\nCan't set to highest RTAPI priority.");


    // Open the required IPC objects upon first call.
    if (!bInit)
    {
        hShm = RtOpenSharedMemory( PAGE_READWRITE, FALSE, DAQ_SHM, (VOID **) &pDaq);
        if (hShm==NULL)
        {
               MsgAndExit("Could not open Shared Memory.");
               return FALSE;
        }

        hSemPost = RtOpenSemaphore( SYNCHRONIZE, FALSE, DAQ_SEM_POST);
        if (hSemPost==NULL)
        {
               MsgAndExit("Could not open Semaphore.");
               //RtCloseHandle(hShm);
               return FALSE;
        }
 }

 pDaq->index = 0;

 // Setup and start a periodic timer.
 if (!(hTimer = RtCreateTimer(NULL,    // Security - NULL is none
                                0,     // Stack size - 0 is use default
                                TimerHandler,      // Timer handler
                                NULL,  // NULL context (argument to handler)
                                RT_PRIORITY_MAX,  // Priority
                                CLOCK_2)))  // Always use fastest available clock
    {
        MsgAndExit("Could not create the timer.");
        return ERROR_OCCURED;
    }
```

```
    if (!RtSetTimerRelative( hTimer, &Period, &Period))
    {
        MsgAndExit("Could not set and start the timer.");
        RtDeleteTimer( hTimer);
        return ERROR_OCCURED;
    }

 SuspendThread(ThreadD);

 if(!RtDeleteTimer( hTimer))
 {
        MsgAndExit("Could not delete timer.");
        return ERROR_OCCURED;
 }
 RtCloseHandle(hShm);
 RtCloseHandle(hSemPost);
 ExitThread(retcode);
 //CloseHandle(ThreadD);
 return 0;
}

void RTFCNDCL TimerHandler (PVOID   unused)
{
 // Test sampling rate

 *PORTC_ADDR = 0x00;

 pDaq->Buffer[pDaq->index] = *PORTA_ADDR;
 pDaq->index++;
 if(pDaq->index == BUFFER_SIZE/2)
 {
        pDaq->HalfBufFull = 1;
        if(!RtReleaseSemaphore(hSemPost, lReleaseCount, NULL))
        {
          MsgAndExit("Could not release semaphore.");
          //return FALSE;
        }
 }

 if(pDaq->index == BUFFER_SIZE)
 {
        pDaq->HalfBufFull = 2;
        if(!RtReleaseSemaphore(hSemPost, lReleaseCount, NULL))
        {
          MsgAndExit("Could not release semaphore.");
          //return FALSE;
        }
   pDaq->index=0;
 }

 *PORTC_ADDR = 0xff;
}


PCHAR SetupDIO96()
{
    ULONG       i;                  // logical slot number for the PCI adapter
    ULONG       f;                  // function number on the specified adapter
    ULONG       bytesWritten;       // return value from RtGetBusDataByOffset
    ULONG       bus;                // bus number
    BOOLEAN     flag;
 ULONG          Offset= 0;
 ULONG          nothingWritten = 0;
 ULONG          AddressSpace = 0;
 ULONG          window_data_value;
 PCI_SLOT_NUMBER     SlotNumber;
 PPCI_COMMON_CONFIG  PciData;
 UCHAR   buffer[PCI_COMMON_HDR_LENGTH];
 PCHAR vBAR0;
 PCHAR vBAR1;
```

```
LARGE_INTEGER BAR0;
LARGE_INTEGER BAR1;
LARGE_INTEGER tBAR0;
LARGE_INTEGER tBAR1;

BAR0.QuadPart = 0;
BAR1.QuadPart = 0;
tBAR0.QuadPart = 0;
tBAR1.QuadPart = 0;

PciData = (PPCI_COMMON_CONFIG) buffer;
   SlotNumber.u.bits.Reserved = 0;
   flag = TRUE;

   for (bus=0; flag; bus++) {

       for (i=0; i<PCI_MAX_DEVICES && flag; i++) {
           SlotNumber.u.bits.DeviceNumber = i;

           for (f=0; f<PCI_MAX_FUNCTION; f++) {
               SlotNumber.u.bits.FunctionNumber = f;

               bytesWritten = RtGetBusDataByOffset (
                   PCIConfiguration,
                   bus,
                   SlotNumber.u.AsULONG,
                   PciData,
                   Offset,
                   PCI_COMMON_HDR_LENGTH
                   );

               if (bytesWritten == nothingWritten) {
                   // out of PCI buses
                   flag = FALSE;
                   break;
               }

               if (PciData->VendorID == PCI_INVALID_VENDORID) {
                   // no device at this slot number, skip to next slot
                   break;
               }

               if((PciData->VendorID == 0x1093) &&(PciData->DeviceID == 0x0160))
               {
                      BAR0.QuadPart = PciData->u.type0.BaseAddresses[0];
                      BAR1.QuadPart = PciData->u.type0.BaseAddresses[1];

                      //Traslate the base port address
                      if(! RtTranslateBusAddress(PCIBus,0,BAR0,&AddressSpace,&tBAR0))
                      {
                             printf("tBAR0 translation failed\n");
                      }
                      else
                      {
                             printf("tBAR0:SystemMappedAddress: 0x%08x\n", tBAR0.LowPart);
                      }

                      if(! RtTranslateBusAddress(PCIBus,0,BAR1,&AddressSpace,&tBAR1))
                      {
                             printf("tBAR1 translation failed\n");
                      }

                      else
                      {
                             printf("tBAR1:SystemMappedAddress: 0x%08x\n", tBAR1.LowPart);
                      }

                      //Map the address to virtual address the software can use
                      vBAR0 = (char*)RtMapMemory(tBAR0, 4*1024,0);
                      if (vBAR0==0) MsgAndExit("Failure on RtmapMemory");
```

```
                        //Setup
                        PciData->Command = (PCI_ENABLE_IO_SPACE|
                                        PCI_ENABLE_MEMORY_SPACE|
                                        PCI_ENABLE_BUS_MASTER|
                                        PCI_ENABLE_WRITE_AND_INVALIDATE);
                        RtSetBusDataByOffset(PCIConfiguration,bus,SlotNumber.u.AsULONG,
                                             PciData,0,PCI_COMMON_HDR_LENGTH);

                        window_data_value = ( (0xffffff00 &
                         (ULONG)BAR1.LowPart) | (0x00000080) );
                        PTL(vBAR0+0x000000c0) = window_data_value;
                        } //dio96
                }
            }
        }
return vBAR1;
}

void MAlloc()
{
 int i;
 dpqueue = (unsigned char **) GlobalAlloc(GPTR, qsize * sizeof(unsigned char *));
 if (dpqueue == NULL )
        MsgAndExit("Couldn't allocate memory");
 for (i=0;i<qsize;i++)
 {
        dpqueue[i] = (unsigned char *)GlobalAlloc(GPTR, BUFFER_SIZE/2 * sizeof(unsigned
char));
        if (dpqueue[i] == NULL )
                MsgAndExit("Couldn't allocate memory");
 }
}
void MFree()
{
 int i;
 if (dpqueue != NULL ) GlobalFree (dpqueue);
 for(i=0;i<qsize;i++)
        if (dpqueue[i] != NULL ) GlobalFree (dpqueue[i]);
}

#ifdef MALLOC
void MAlloc()
{
 int i;
 /*hpt1 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt2 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt3 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt4 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt5 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt6 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt7 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));
 hpt8 = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(int));*/
 hqueue = GlobalAlloc(GMEM_MOVEABLE, qsize * BUFFER_SIZE/2 * sizeof(unsigned char));
 /*for(i=0;i<qsize;i++)
        hqueue[i] = GlobalAlloc(GMEM_MOVEABLE, BUFFER_SIZE/2 * sizeof(char));*/

 if ((hqueue == NULL))
   {
      //sprintf(szTemp,"Couldn't allocate memory");
      MFree();
      return ;
   }

 /*pt1 = (int *)GlobalLock(hpt1);
 pt2 = (int *)GlobalLock(hpt2);
 pt3 = (int *)GlobalLock(hpt3);
 pt4 = (int *)GlobalLock(hpt4);
 pt5 = (int *)GlobalLock(hpt5);
 pt6 = (int *)GlobalLock(hpt6);
 pt7 = (int *)GlobalLock(hpt7);
 pt8 = (int *)GlobalLock(hpt8);*/
```

```
    dpqueue = (unsigned char **)GlobalLock(hqueue);
 for(i=0;i<qsize;i++)
        dpqueue[i] = (unsigned char *)Globalloc(hqueue[i]);

 if (dpqueue == NULL)
   {
      //sprintf(szTemp,"Couldn't allocate memory");
      MFree();
      return ;
   }

}

void MFree()
{

 /*if (pt1 != NULL)
 {
        GlobalUnlock(pt1);
        pt1 = NULL;
 }*/

 if (dpqueue != NULL)
 {
        GlobalUnlock(dpqueue);
        dpqueue = NULL;
 }

 /*if (hpt1 != NULL)
 {
        GlobalFree(hpt1);
        hpt1 = NULL;
 }*/

 if (hqueue != NULL)
 {
        GlobalFree(hqueue);
        hqueue = NULL;
 }

}
#endif

void Plot()
{
 HBRUSH  hbrush[8], hbrushb1;
 HPEN    hPen1, hPen2, hPen3, hPen4, hPen5, hPen6, hPen7,hPen8, hOldPen;
 RECT    rect,rect1[8];
 char    szbuffer[100];
 //static char  pointLast[8];
 int            iLength, i;


                hdc = BeginPaint(hwnd, &ps);
                hbrush[0] = CreateSolidBrush(RGB(255, 0, 0));
                hbrush[1] = CreateSolidBrush(RGB(0, 255, 0));
                hbrush[2] = CreateSolidBrush(RGB(0, 0, 255));
                hbrush[3] = CreateSolidBrush(RGB(250, 150, 0));
                hbrush[4] = CreateSolidBrush(RGB(0, 255, 255));
                hbrush[5] = CreateSolidBrush(RGB(255, 0, 255));
                hbrush[6] = CreateSolidBrush(RGB(255, 255, 0));
                hbrush[7] = CreateSolidBrush(RGB(0, 0, 0));
                hbrushb1 = CreateSolidBrush(RGB(200, 200, 200));
                hPen6 = CreatePen(PS_SOLID, 2, RGB(100, 100, 100));
                hPen4 = CreatePen(PS_SOLID, 2, RGB(150, 150, 150));
                hOldPen = (HPEN) SelectObject(hdc, hPen6);
                rect.left = 0 ;
                rect.right = 760 ;
                rect.top = 2 ;
                rect.bottom = 512 ;
                FillRect(hdc, &rect, hbrushb1);
```

```
SelectObject (hdc,GetStockObject(WHITE_BRUSH));
Rectangle (hdc, 160, 60, 720, 230);          //Curve Rect
Rectangle (hdc, 25, 60, 120, 230);           //Child Rect
Rectangle (hdc, 15, 270, 185, 475);          //Pair-1 Rect
Rectangle (hdc, 200, 270, 370, 475);  //Pair-2 Rect
Rectangle (hdc, 385, 270, 555, 475);  //Pair-3 Rect
Rectangle (hdc, 570, 270, 740, 475);  //Pair-4 Rect

for(i=0;i<8;i++)
{
       rect1[i].left = 30 ;
       rect1[i].right = 43;
       rect1[i].top = 68 + 20*i;
       rect1[i].bottom = 80 + 20*i;
       FillRect(hdc, &rect1[i], hbrush[i]);
}

SelectObject(hdc, hPen4);
MoveToEx (hdc, 0, 40, NULL);
LineTo (hdc, 760, 40);
SelectObject (hdc,hPen6);
MoveToEx (hdc, 0, 258, NULL);
LineTo (hdc, 760, 258);
MoveToEx (hdc, 0, 482, NULL);
LineTo (hdc, 760, 482);
MoveToEx (hdc, 193, 275, NULL);
LineTo (hdc, 193, 465);
MoveToEx (hdc, 378, 275, NULL);
LineTo (hdc, 378, 465);
MoveToEx (hdc, 563, 275, NULL);
LineTo (hdc, 563, 465);
SelectObject (hdc,GetStockObject(WHITE_PEN));
MoveToEx (hdc, 0, 255, NULL);
LineTo (hdc, 760, 255);
MoveToEx (hdc, 0, 485, NULL);
LineTo (hdc, 760, 485);
MoveToEx (hdc, 190, 275, NULL);
LineTo (hdc, 190, 465);
MoveToEx (hdc, 375, 275, NULL);
LineTo (hdc, 375, 465);
MoveToEx (hdc, 560, 275, NULL);
LineTo (hdc, 560, 465);
MoveToEx (hdc, 718, 61, NULL);
LineTo (hdc, 718, 228);
LineTo (hdc, 160, 228);

SetBkMode (hdc, TRANSPARENT); //Ignore the background color
SelectObject (hdc,hPen6);
for (i=0; i<4; i++)
{
       rect.left = 18 + 185 * i ;
       rect.right = 182 + 185 * i;
       rect.top = 273 ;
       rect.bottom = 298 ;
       FillRect(hdc, &rect, hbrushb1);
       MoveToEx (hdc, 18 + 185 * i, 298, NULL);
       LineTo (hdc, 181 + 185 * i, 298);
       iLength=wsprintf(szbuffer,"Pair-%d",i+1);
       TextOut(hdc,73+185*i,275,szbuffer,iLength);
       iLength=sprintf(szbuffer,"v1= %-6.2f",v[i][0]);
       TextOut(hdc,25+185*i,320,szbuffer,iLength);
       iLength=sprintf(szbuffer,"(mile/h)");
       TextOut(hdc,120+185*i,320,szbuffer,iLength);
       iLength=sprintf(szbuffer,"v2= %-6.2f",v[i][1]);
       TextOut(hdc,25+185*i,370,szbuffer,iLength);
       iLength=sprintf(szbuffer,"(mile/h)");
       TextOut(hdc,120+185*i,370,szbuffer,iLength);
       //iLength=sprintf(szbuffer,"a  = %-6.2f",a[i]);
       //TextOut(hdc,25+185*i,390,szbuffer,iLength);
       //iLength=sprintf(szbuffer,"(mi/h^2)");
```

```
            //TextOut(hdc,120+185*i,390,szbuffer,iLength);
            iLength=sprintf(szbuffer,"L  = %-6.2f",l[i]);
            TextOut(hdc,25+185*i,420,szbuffer,iLength);
            iLength=sprintf(szbuffer,"( m )");
            TextOut(hdc,120+185*i,420,szbuffer,iLength);
      }

for(i=0;i<8;i++)
      DeleteObject(hbrush[i]);
      DeleteObject(hbrushb1);
      SelectObject(hdc, hOldPen);
      DeleteObject(hPen6);
      DeleteObject(hPen4);

      SelectObject(hdc,hrgnu);      //Curve

if ( Pen1 )
      hPen1 = CreatePen(PS_SOLID, 3, RGB(255, 0, 0));
else
      hPen1 = CreatePen(PS_NULL, 3, RGB(255, 0, 0));
if ( Pen2 )
      hPen2 = CreatePen(PS_SOLID, 3, RGB(0, 255, 0));
else
      hPen2 = CreatePen(PS_NULL, 3, RGB(0, 255, 0));
if ( Pen3 )
      hPen3 = CreatePen(PS_SOLID, 3, RGB(0, 0, 255));
else
      hPen3 = CreatePen(PS_NULL, 3, RGB(0, 0, 255));
if ( Pen4 )
      hPen4 = CreatePen(PS_SOLID, 3, RGB(250, 150, 0));
else
      hPen4 = CreatePen(PS_NULL, 3, RGB(250, 150, 0));
if ( Pen5 )
      hPen5 = CreatePen(PS_SOLID, 3, RGB(0, 255, 255));
else
      hPen5 = CreatePen(PS_NULL, 3, RGB(0, 255, 255));
if ( Pen6 )
      hPen6 = CreatePen(PS_SOLID, 3, RGB(255, 0, 255));
else
      hPen6 = CreatePen(PS_NULL, 3, RGB(255, 0, 255));
if ( Pen7 )
      hPen7 = CreatePen(PS_SOLID, 3, RGB(255, 255, 0));
else
      hPen7 = CreatePen(PS_NULL, 3, RGB(255, 255, 0));
if ( Pen8 )
      hPen8 = CreatePen(PS_SOLID, 3, RGB(0, 0, 0));
else
      hPen8 = CreatePen(PS_NULL, 3, RGB(0, 0, 0));
      hOldPen = (HPEN) SelectObject(hdc, hPen1);
if (exmove)
{
      MoveToEx(hdc, 710-1, pointLast[0], NULL);
      LineTo(hdc, 710, point1[0]);
      SelectObject(hdc, hPen2);
      MoveToEx(hdc, 710-1, pointLast[1], NULL);
      LineTo(hdc, 710, point2[0]);
      SelectObject(hdc, hPen3);
      MoveToEx(hdc, 710-1, pointLast[2], NULL);
      LineTo(hdc, 710, point3[0]);
      SelectObject(hdc, hPen4);
      MoveToEx(hdc, 710-1, pointLast[3], NULL);
      LineTo(hdc, 710, point4[0]);
      SelectObject(hdc, hPen5);
      MoveToEx(hdc, 710-1, pointLast[4], NULL);
      LineTo(hdc, 710, point5[0]);
      SelectObject(hdc, hPen6);
      MoveToEx(hdc, 710-1, pointLast[5], NULL);
      LineTo(hdc, 710, point6[0]);
      SelectObject(hdc, hPen7);
      MoveToEx(hdc, 710, pointLast[6], NULL);
      LineTo(hdc, 710-1, point7[0]);
```

```
      SelectObject(hdc, hPen8);
      MoveToEx(hdc, 710-1, pointLast[7], NULL);
      LineTo(hdc, 710, point8[0]);
      exmove = FALSE;
}
else
{
      MoveToEx(hdc, 710-DX, point1[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point1[i+Lastpoint+1]);
      SelectObject(hdc, hPen2);
      MoveToEx(hdc, 710-DX, point2[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point2[i+Lastpoint+1]);
      SelectObject(hdc, hPen3);
      MoveToEx(hdc, 710-DX, point3[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point3[i+Lastpoint+1]);
      SelectObject(hdc, hPen4);
      MoveToEx(hdc, 710-DX, point4[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point4[i+Lastpoint+1]);
      SelectObject(hdc, hPen5);
      MoveToEx(hdc, 710-DX, point5[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point5[i+Lastpoint+1]);
      SelectObject(hdc, hPen6);
      MoveToEx(hdc, 710-DX, point6[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point6[i+Lastpoint+1]);
      SelectObject(hdc, hPen7);
      MoveToEx(hdc, 710-DX, point7[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point7[i+Lastpoint+1]);
      SelectObject(hdc, hPen8);
      MoveToEx(hdc, 710-DX, point8[Lastpoint], NULL);
      for(i=0; i<DX; i++)
        LineTo(hdc, i+710-DX+1, point8[i+Lastpoint+1]);
      Lastpoint = Lastpoint + DX;
}
      if (Lastpoint == BUFFER_SIZE/2-1)
      {
             pointLast[0]=point1[BUFFER_SIZE/2-1];
             pointLast[1]=point2[BUFFER_SIZE/2-1];
             pointLast[2]=point3[BUFFER_SIZE/2-1];
             pointLast[3]=point4[BUFFER_SIZE/2-1];
             pointLast[4]=point5[BUFFER_SIZE/2-1];
             pointLast[5]=point6[BUFFER_SIZE/2-1];
             pointLast[6]=point7[BUFFER_SIZE/2-1];
             pointLast[7]=point8[BUFFER_SIZE/2-1];
      }

      SelectObject(hdc, hOldPen);
      DeleteObject(hPen1);
      DeleteObject(hPen2);
      DeleteObject(hPen3);
      DeleteObject(hPen4);
      DeleteObject(hPen5);
      DeleteObject(hPen6);
      DeleteObject(hPen7);
      DeleteObject(hPen8);

      SelectObject(hdc,hrgnd);      //v1,v2,a,l
      for (i=0; i<4; i++)
      {
             iLength=sprintf(szbuffer,"%-6.2f",v[i][0]);
             TextOut(hdc,55+185*i,320,szbuffer,iLength);
             iLength=sprintf(szbuffer,"%-6.2f",v[i][1]);
             TextOut(hdc,55+185*i,370,szbuffer,iLength);
             //sprintf(szbuffer,"%-6.2f",a[i]);
             //TextOut(hdc,55+185*i,390,szbuffer,6);
```

```
                    iLength=sprintf(szbuffer,"%-6.2f",l[i]);
                    TextOut(hdc,55+185*i,420,szbuffer,iLength);
                }
                EndPaint(hwnd, &ps);
}

void GetFileName()
{
 char szFilter[256], chReplace;
 //static char szFile[256]="", szFileTitle[256]="";
 UINT i,cbString;
 static OPENFILENAME ofn;

 strcpy(szFilter,
         "All files (*.*)|*.*|"
         "Header files (*.h)|*.h|"
         "C++ (*.cpp)|*.cpp|"
                 "Data files (*.dat)|*.dat||");

         cbString = strlen(szFilter);
         chReplace = szFilter[cbString-1];
         for (i=0; szFilter[i]; i++)
            if (szFilter[i] == chReplace) szFilter[i] = '\0';

         memset(&ofn, 0, sizeof(OPENFILENAME));
         ofn.lStructSize = sizeof(OPENFILENAME);
         ofn.hwndOwner = hwnd;
         ofn.lpstrFilter = szFilter;
         ofn.nFilterIndex = 4;
         ofn.lpstrFile = szFile;
         ofn.nMaxFile = sizeof(szFile);
         ofn.lpstrFileTitle = szFileTitle;
         ofn.nMaxFileTitle = sizeof(szFileTitle);
         ofn.lpstrInitialDir = NULL; // szDirName;
         GetOpenFileName(&ofn);
          //filename = szFileTitle;
}

/*******************************
 * sensorOpen()                                 *
 *                                                        *
 * Open sensor communication   *
 * return sensor instance              *
 *******************************/
filesys_t* sensorOpen( )
{
    //int i;
    filesys_t   *s;     /* file instance */

 /* allocate sensor */
    /*s = (filesys_t*)calloc(1,sizeof(filesys_t));
    if (!s)
        printf("sensorOpen(): calloc() error\n"), exit(1);*/

 s = (filesys_t *)GlobalAlloc(GPTR, sizeof(filesys_t));
 if (s == NULL )
        MsgAndExit("Couldn't allocate filesys_t memory");

                                                /* init sensor */
    //s->queue = (short***)NULL;
    //s->tick = (clock_t*)NULL;
    s->status = 0;
    //s->qsize = 0;

    return s;
}

/*******************************
 * sensorClose()                                *
 *                                                        *
 * close sensor communication  *
```

```
 ******************************/
void sensorClose( filesys_t* s )
{
    //int i;

    debug(("sensorClose() {\n"));

    if (!s) return;
                                                /* clear bits */
    s->status &= ~(PLAY|RECORD);

    if (s->file)
 fclose(s->file);

 s->file = (FILE*)NULL;

                                        /* free memory */
    /*for (i=0; i<s->qsize; i++)
        free(s->queue[i]);

    if (s->queue)
        free(s->queue);

    if (s->tick)
        free(s->tick);

    free(s);

    memset(s, 0x0, sizeof(filesys_t));*/

 if (s != NULL ) GlobalFree (s);

    debug(("} sensorClose()\n"));
}

/* ----------------------- File Record/Play ----------------------- */

/**************************************
 * FileOpen()                                                    *
 *                                                                                  *
 * open filename for record/play              *
 **************************************/
int FileOpen( filesys_t *s, char *szFileTitle )
{
    int file_status;
    //header_t header;

    if (!s || !szFileTitle)
 return -1;

                               /* check if file exists */
    errno = 0;
    file_status = _access(szFileTitle, 0);
    debug(("\tfile_satus: %d\n",file_status));

    /*if (errno && errno!=ENOENT) {
 perror("access()");
 return -1;
    }*/

    if (file_status==0) {
                               /* file exists -> read */

 s->file = fopen(szFileTitle, "rb+");
 if (!s->file) {
     perror(szFileTitle);
     return -1;
 }
                               /* read header */

 /*fread( &header, sizeof(header_t), 1, s->file );
```

```
 if (ferror(s->file)) {
     perror(filename);
     return -1;
 }
 fflush(s->file);*/
                                /* check if same as sensor */

 /*if (header.n_diode != N_DIODE) {
     fprintf(stderr,"%s: %d, %d: N_DIODE mismatch\n",
             filename, header.n_diode, N_DIODE);
     return -1;
 }*/

 /*if (header.n_scan != s->n_scan) {
     fprintf(stderr,"%s: %d, %d: n_scan mismatch\n",
             filename, header.n_scan, s->n_scan);
     return -1;
 }*/
  } else {
                                /* no file exists -> write */
 NoPlay = TRUE;
 debug(("\tfopen(%s,\"wb\")\n",szFileTitle));
 s->file = fopen(szFileTitle, "wb");

 if (!s->file) {
     perror(szFileTitle);
     return -1;
 }
                                /* write header */
 //header.n_diode = N_DIODE;
 //header.n_scan = s->n_scan;

/*#ifdef FILE_DATA_TIME
 header.time = s->start_time;
#endif

 fwrite( &header, sizeof(header_t), 1, s->file );
 if (ferror(s->file)) {
     perror(filename);
     return -1;
 }
 fflush(s->file);*/
  }

    s->block_size = BUFFER_SIZE/2;
    if((s->curr_pos = ftell(s->file))==-1L)
        MsgAndExit("ftell error.");;
    s->data_pos = s->curr_pos;

    if (fseek(s->file, s->data_pos, SEEK_SET)<0) {
 perror("FileOpen(): fseek()");
 return -1;
    }
                                                /* playback timeout */

    //s->play_timeout = (long)( s->n_scan * s->period * 1e6 );

    return 0;
}


/*******************************
 * FileClose()                                  *
 *                                                      *
 * close record/play file              *
 *******************************/
int FileClose( filesys_t *s )
{
    debug(("FileClose() {"));

    if (!s)
```

```
 return -1;
                              /* stop record/play */
    s->status &= ~(RECORD|PLAY);

    debug(("} FileClose()\n"));

                                      /* close file if open */
    if (s->file)
 fclose(s->file);

    s->file = (FILE*)NULL;

    return 0;
}


/*****************************************************
 * FileBegin()                                                                  *
 *
       *
 * move read/write position to beginning of file             *
 *****************************************************/
int FileBegin( filesys_t *s )
{
    debug(("FileBegin() {"));

    if (!s || !s->file)
 return -1;
                              /* stop record/play */
    s->status &= ~(RECORD|PLAY);
                                      /* move to starting position */
    fflush(s->file);
    if (fseek(s->file,s->data_pos,SEEK_SET)<0) {
 perror("FileBegin(): fseek()");
 return -1;
    }
                                      /* get position */
    s->curr_pos = ftell(s->file);

    debug(("} FileBegin()\n"));

    return 0;
}


/*********************************************
 * FileEnd()                                                          *
 *                                                                              *
 * move read/write position to end of file           *
 *********************************************/
int FileEnd( filesys_t *s )
{
    debug(("FileEnd() {"));

    if (!s || !s->file)
 return -1;
                              /* stop record/play */
    s->status &= ~(RECORD|PLAY);
                              /* move to end of file */
    fflush(s->file);
    if (fseek(s->file,0L,SEEK_END)<0) {
 perror("FileEnd(): fseek()");
 return -1;
    }
                                      /* get position */
    s->curr_pos = ftell(s->file);

    debug(("} FileEnd()\n"));

    return 0;
}
```

```
/*************************************
 * FileBack()                                                    *
 *                                                                                  *
 * move read/write position backwards  *
 * n data blocks                                                 *
 *************************************/
int FileBack( filesys_t *s, int n )
{
    long offset;
    int mode = SEEK_CUR;

    debug(("FileBack() {"));

    if (!s || !s->file || n<=0)
 return -1;

    fflush(s->file);
                                /* get position */
    s->curr_pos = ftell(s->file);
                                /* check if beginning */
    if (s->curr_pos==s->data_pos)
 return 0;
                                        /* move n data blocks */
    offset = n * s->block_size;

                                        /* check if too far */
    if ((s->curr_pos - offset) <= s->data_pos) {
                                        /* set to beginning */
 offset = s->data_pos;
 mode = SEEK_SET;

    } else offset = -offset;

                                        /* move backwards */
    if (fseek(s->file,offset,mode)<0) {
 perror("FileBack(): fseek()");
 return -1;
    }
                                        /* get position */
    s->curr_pos = ftell(s->file);

    debug(("} FileBack()\n"));

    return 0;
}


/*************************************
 * FileForward()                                                 *
 *                                                                                  *
 * move read/write position forward           *
 * n data blocks                                                 *
 *************************************/
int FileForward( filesys_t *s, int n )
{
    long offset;

    debug(("FileForward() {"));

    if (!s || !s->file || n<=0)
 return -1;

    fflush(s->file);

                                        /* move n data blocks */
    offset = n * s->block_size;

                                        /* move forwards */
    if (fseek(s->file,offset,SEEK_CUR)<0) {
```

```
 perror("FileForward(): fseek()");
 return -1;
    }
                                      /* get position */
    s->curr_pos = ftell(s->file);

    debug(("} FileForward()\n"));

    return 0;
}


/************************
 * FileStop()                  *
 *                                        *
 * stop record/play            *
 ************************/
int FileStop( filesys_t *s )
{
    if (FilePause(s)<0)
 return -1;

    Sleep(1);

    if (FileBegin(s)<0)
 return -1;

    return 0;
}


/************************
 * FilePause()                 *
 ************************/
int FilePause( filesys_t *s )
{
    debug(("FilePause() {"));
    if (!s)
 return -1;

    s->status &= ~(RECORD|PLAY);
 s->status |= PAUSE;

    if (s->file) {
 //fflush(s->file);

 s->curr_pos = ftell(s->file);
    }

    debug(("} FilePause()\n"));
    return 0;
}


/*******************************
 * FilePlayEOF()                           *
 *                                                  *
 * check if at End Of File             *
 *******************************/
int FilePlayEOF( filesys_t *s )
{
    if (!s)
 return -1;

    if (s->status & PLAY_EOF)
 return 1;
    else
 return 0;
}
```

```
/*******************************
 * FilePosition()                              *
 *                                                          *
 * return current file position       *
 *******************************/
long FilePosition( register filesys_t *s )
{
    if (!s || !s->file)
 return -1;

    return (s->curr_pos = ftell(s->file));
}


/*******************************
 * FileSetPlayTimeout()              *
 *******************************/
int FileSetPlayTimeout( filesys_t *s, long timeout )
{
    if (!s || timeout<=0)
 return -1;

    s->play_timeout = timeout;

    return 0;
}


/*******************************
 * FileGetPlayTimeout()              *
 *******************************/
long FileGetPlayTimeout( filesys_t *s )
{
    if (!s)
 return -1;

    return s->play_timeout;
}

/* ---------------------------- Record ----------------------------- */


/*******************************
 * FileRecord()                                *
 *                                                          *
 * start recording to file            *
 *******************************/
int FileRecord( filesys_t *s )
{
    debug(("FileRecord() {"));
    if (!s || !s->file || (s->status & PLAY))
 return -1;

    s->status |= RECORD;

    debug(("} FileRecord()\n"));
    return 0;
}



/* ---------------------------- Play ----------------------------- */



/************************
 * FilePlay()                   *
 *                                                *
 * start playing file   *
 ************************/
int FilePlay( filesys_t *s )
```

```
{
    debug(("FilePlay() {"));

    if (!s || !s->file || (s->status & RECORD))
 return -1;

    s->status &= ~(PLAY_EOF);
    s->status |= PLAY;

    debug(("} FilePlay()\n"));
    return 0;
}

/***************************
 * FileCont()                          *
 *                                                    *
 * continue playing file       *
 ***************************/
int FileCont( filesys_t *s )
{
    debug(("FilePlay() {"));

    if (!s || !s->file || (s->status & RECORD))
 return -1;

    s->status &= ~(PLAY_EOF);
 s->status &= ~PAUSE;
    s->status |= PLAY;

    debug(("} FilePlay()\n"));
    return 0;
}


void DaqStart()
{
 InvalidateRgn (hwnd, hrgnu, TRUE) ;
 if (SetTimer (hwnd, 1, 10, NULL))
    {
         EnableMenuItem (hMenu, IDM_DAQSTART, MF_GRAYED) ;
         EnableMenuItem (hMenu, IDM_DAQEND,  MF_ENABLED) ;
    }
 Lastpoint = 0;
 bkill=FALSE;
 MAlloc();
 ThreadU=CreateThread(NULL,0,Thread96U,NULL,0,NULL);
 ThreadD=CreateThread(NULL,0,Thread96D,NULL,0,NULL);
 DAQ = TRUE;
}

void DaqEnd()
{
 KillTimer (hwnd, 1) ;
 EnableMenuItem (hMenu, IDM_DAQSTART, MF_ENABLED) ;
    EnableMenuItem (hMenu, IDM_DAQEND,  MF_GRAYED) ;
 bkill=TRUE;
 DAQ = FALSE;
}

void Record()
{
 FileClose( s );
 FileOpen(s,szFileTitle);
 NoPlay=FALSE;
 FileRecord( s );
 if ( !DAQ ) DaqStart();
}

void RPStop()
{
 if(s->status & PLAY)
```

```
 {
        FileClose( s );
        KillTimer (hwnd, 1) ;
 }
 if(s->status & RECORD)
 {
        DaqEnd();
        FileClose( s );
 }
}

void Play()
{
 int i;
 for(i=0; i<N_PAIR; i++)
 {
        v[i][0] = 0;
        v[i][1] = 0;
        l[i] = 0;
 }
 InvalidateRgn (hwnd, hrgnu, TRUE) ;
 InvalidateRgn (hwnd, hrgnd, TRUE) ;
 FileClose( s );
 FileOpen(s,szFileTitle);
 if(!NoPlay)
 {
        FilePlay( s );
        for(i=0; i<BUFFER_SIZE/2; i++)
        {
               point1[i] = 100;
               point2[i] = 100;
               point3[i] = 100;
               point4[i] = 100;
               point5[i] = 100;
               point6[i] = 100;
               point7[i] = 100;
               point8[i] = 100;
        }
        SetTimer (hwnd, 1, 10, NULL);
        Lastpoint = 0;
        iFile = 0;
 }
 else   MsgAndExit("File Dosen't Exsist.");
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
     //HMENU       hMenu ;                          //handle of menu
  static HWND hwndButton[8];                  //handle of Child window
  TEXTMETRIC   tm ;
  static int   cxChar, cyChar ;

  int i,j,k;
  static char old = 0xFF;                                 //the last data
  static unsigned long edge_status[N_PAIR];   //status of edge, xxxxT3T2T1T0
  static int e[N_PAIR];                                   //the order of edge, 0,1,2,3
  static unsigned long t[N_PAIR][4];          //edge timing information
  /*enum{                                                          //vehicle direction
               None,
               Presence,
               Trigger
  } edge[N_PAIR];*/
  static double delta_t,              //dt
                       delta_v;               //dv

  static HINSTANCE hInstance ;
  static WNDPROC   lpfnAboutDlgProc ;

     switch (iMsg)
          {
```

```
case WM_CREATE:
      hdc = GetDC (hwnd) ;
       SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
       GetTextMetrics (hdc, &tm) ;
       cxChar = tm.tmAveCharWidth ;
       cyChar = tm.tmHeight + tm.tmExternalLeading ;
       ReleaseDC (hwnd, hdc) ;

       for (i = 0 ; i < 8 ; i++)             //Child window
            hwndButton[i] = CreateWindow ("button", button[i].text,
                      WS_CHILD | WS_VISIBLE | button[i].style,
                      5 * cxChar, cyChar * (3.1 + i),
                      6 * cxChar, 5 * cyChar / 4,
                      hwnd, (HMENU) (200+i),
                      ((LPCREATESTRUCT) lParam) -> hInstance, NULL) ;

              hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
        return 0 ;

case WM_PAINT:
        Plot();
        break;

case WM_COMMAND :
       hMenu = GetMenu (hwnd) ;

       switch (LOWORD (wParam))
            {
            case IDM_CH1 :
                              if ( Pen1 ) Pen1 = FALSE;
                              else  Pen1 = TRUE;
                              return 0;

            case IDM_CH2 :
                              if ( Pen2 ) Pen2 = FALSE;
                              else  Pen2 = TRUE;
                              return 0;

            case IDM_CH3 :
                              if ( Pen3 ) Pen3 = FALSE;
                              else  Pen3 = TRUE;
                              return 0;

            case IDM_CH4 :
                              if ( Pen4 ) Pen4 = FALSE;
                              else  Pen4 = TRUE;
                              return 0;

            case IDM_CH5 :
                              if ( Pen5 ) Pen5 = FALSE;
                              else  Pen5 = TRUE;
                              return 0;

            case IDM_CH6 :
                              if ( Pen6 ) Pen6 = FALSE;
                              else  Pen6 = TRUE;
                              return 0;

            case IDM_CH7 :
                              if ( Pen7 ) Pen7 = FALSE;
                              else  Pen7 = TRUE;
                              return 0;

            case IDM_CH8 :
                              if ( Pen8 ) Pen8 = FALSE;
                              else  Pen8 = TRUE;
                              return 0;

            case IDM_RECORDSTART :
                              /*FileClose( s );
                              FileOpen(s,szFileTitle);
```

```
                        NoPlay=FALSE;
                        FileRecord( s );
                        if ( !DAQ ) DaqStart();*/
                        Record();
                        return 0;

                case IDM_RECORDEND :
                                /*DaqEnd();
                        FileClose( s );*/
                        RPStop();
                     return 0 ;

                case IDM_PLAYSTART :
                        /*for(i=0; i<N_PAIR; i++)
                        {
                                v[i][0] = 0;
                                v[i][1] = 0;
                                l[i] = 0;
                        }
                        InvalidateRgn (hwnd, hrgnu, TRUE) ;
                        InvalidateRgn (hwnd, hrgnd, TRUE) ;
                        FileClose( s );
                        FileOpen(s,szFileTitle);
                        if(!NoPlay)
                        {
                               FilePlay( s );
                               for(i=0; i<BUFFER_SIZE/2; i++)
                               {
                                      point1[i] = 100;
                                      point2[i] = 100;
                                      point3[i] = 100;
                                      point4[i] = 100;
                                      point5[i] = 100;
                                      point6[i] = 100;
                                      point7[i] = 100;
                                      point8[i] = 100;
                               }
                               SetTimer (hwnd, 1, 10, NULL);
                               Lastpoint = 0;
                               iFile = 0;
                        }
                        else   MsgAndExit("File Dosen't Exsist.");*/
                        Play();
                        return 0;

                case IDM_PLAYEND :
                        /*FileClose( s );
                        KillTimer (hwnd, 1) ;*/
                        RPStop();
                        return 0;

                case IDM_EXIT :
                     SendMessage (hwnd, WM_CLOSE, 0, 0L) ;
                     return 0 ;

                /*case IDM_CHAN1 :
                        if ( Pen1 )
                        { Pen1 = FALSE;
                          CheckMenuItem (hMenu, LOWORD (wParam), MF_UNCHECKED)
;
                        }
                        else
                        { Pen1 = TRUE;
                          CheckMenuItem (hMenu, LOWORD (wParam), MF_CHECKED) ;
                        }
                        return 0;*/

                case IDM_RP :
                     /*if (SetTimer (hwnd, 1, 30, NULL))
                     {
                          EnableMenuItem (hMenu, IDM_START, MF_GRAYED) ;
```

```
                         EnableMenuItem (hMenu, IDM_STOP,  MF_ENABLED) ;
                    }*/
                                hDlgModeless=CreateDialog (hInstance, "DDDIALOGBAR",
hwnd, AboutDlgProc) ;
                                ShowWindow (hDlgModeless,SW_SHOW);
                    return 0 ;

               case IDM_STOP :
                    /*KillTimer (hwnd, 1) ;
                    EnableMenuItem (hMenu, IDM_START, MF_ENABLED) ;
                    EnableMenuItem (hMenu, IDM_STOP,  MF_GRAYED) ;*/
                    return 0 ;

               case IDM_DAQSTART:
                                if ( !DAQ ) DaqStart();
                                return 0;

                         case IDM_DAQEND :
                                if ( DAQ && !(s->status & RECORD )) DaqEnd();
                                return 0 ;

                         case IDM_HELP :
                    MessageBox (hwnd, "Help not yet implemented!",
                              szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    return 0 ;

               case IDM_ABOUT :
                    MessageBox (hwnd, "Menu Demonstration Program.",
                               szAppName, MB_ICONINFORMATION | MB_OK) ;
                    return 0 ;
               }
          break ;


     case WM_SIZE :
                                        //ScrollWindow Rect
          RePaint.left = 180;
          RePaint.right = 710;
          RePaint.top = 70;
          RePaint.bottom = 220;

          if ( hrgnu )
               DeleteObject ( hrgnu );
          hrgnu = CreateRectRgn(160,60,720,230);    //Curve Region
          if ( hrgnd )
               DeleteObject ( hrgnd );
          hrgnd1 = CreateRectRgn(55,320,117,450);   //data Region of Pair-1
          hrgnd2 = CreateRectRgn(240,320,302,450);  //data Region of Pair-2
          hrgnd3 = CreateRectRgn(425,320,487,450);  //data Region of Pair-3
          hrgnd4 = CreateRectRgn(610,320,672,450);  //data Region of Pair-4
          hrgnd = CreateRectRgn(0, 0, 1, 1);
          hrgnd5 = CreateRectRgn(0, 0, 1, 1);
          hrgnd6 = CreateRectRgn(0, 0, 1, 1);
          CombineRgn (hrgnd5, hrgnd1, hrgnd2, RGN_OR);
          CombineRgn (hrgnd6, hrgnd3, hrgnd4, RGN_OR);
          CombineRgn (hrgnd, hrgnd5, hrgnd6, RGN_OR);      //Pair-1 -- Pair4 data
Region

          //InvalidateRect (hwnd, NULL, TRUE) ;
        return 0 ;

     /*case WM_USER+1 :
          hdc=GetDC(hwnd);
          iLength=sprintf(szbuffer,"%d Half buffers acquired.",iLoopCount);
          TextOut(hdc,10,30+20*iLoopCount,szbuffer,iLength);
         ReleaseDC(hwnd,hdc);
          return 0;*/

     case WM_TIMER :
          if(!(s->status & PAUSE))
          {
```

```
                        if ((Lastpoint==BUFFER_SIZE/2-1) & !(s->status & PLAY_EOF))
                        {
                                if (s->status & PLAY) {              //read from file
                                        //struct timespec rqtp;              //for nanosleep()
                                        if (feof(s->file)) {          //check if end of file
                                                s->status &= ~(PLAY);
                                                s->status |= PLAY_EOF;
                                        }
                                        else {

                                                fread(onebuf, sizeof(unsigned char),
BUFFER_SIZE/2, s->file);
                                                iFile++;
                                                if (ferror(s->file))
                                                        //MsgAndExit("File Read Error.");
                                                perror("File Read Error: fread()"), exit(1);
                                                /*rqtp.tv_sec = 0L;
                                                rqtp.tv_nsec = s->play_timeout;
                                                if (nanosleep(&rqtp,(struct timespec*)NULL)<0)
                                                perror("Thread(): nanosleep()");*/
                                        }
                                }
                                else
                                {
                                        for(i=0;i<BUFFER_SIZE/2;i++)
                                                onebuf[i]=dpqueue[qread][i];
                                }

                                                //for drawing curve
                                exmove = TRUE;
                                Lastpoint = 0;
                                for(i=0; i<BUFFER_SIZE/2; i++)
                                {
                                if( onebuf[i]& 0x01 )
                                        point1[i] = 100;
                                else point1[i] = 190;
                                if(onebuf[i] & 0x02 )
                                        point2[i] = 100;
                                else point2[i] = 190;
                                if(onebuf[i] & 0x04 )
                                        point3[i] = 100;
                                else point3[i] = 190;
                                if(onebuf[i] & 0x08 )
                                        point4[i] = 100;
                                else point4[i] = 190;
                                if(onebuf[i] & 0x10 )
                                        point5[i] = 100;
                                else point5[i] = 190;
                                if(onebuf[i] & 0x20 )
                                        point6[i] = 100;
                                else point6[i] = 190;
                                if(onebuf[i] & 0x40 )
                                        point7[i] = 100;
                                else point7[i] = 190;
                                if(onebuf[i] & 0x80 )
                                        point8[i] = 100;
                                else point8[i] = 190;
                                }

                                //for calculate v,a,l

                                for (i=0; i<BUFFER_SIZE/2; i++) {
                                 for (j=0, k=N_PAIR; j<N_PAIR && k<N_DIODE; j++, k++) {

                                         //array1 leading edge

                                  if ((old&(1L<<j)) && !(onebuf[i]&(1L<<j))) {
                                   if ((edge_status[j]&T0)&&(edge_status[j]&T1))    //spike
                                   {
                                        edge_status[j] &=~(T2|T3);
                                   }
```

```
                                 else {
                                      if ((edge_status[j]&T0) && edge[j]==Trigger)
                                      {
                                             e[j] = 1;                      //second edge
                                             edge_status[j] |= T1;
                                      }
                                      else
                                      {
                                             e[j] = 0;                      //first edge
                                             edge_status[j] = T0;
                                             edge[j]=Presence;
                                      }
                                      if (s->status & PLAY) t[j][e[j]]=((iFile-
1)*BUFFER_SIZE/2+i);
                                      else t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                 }
                                }

                                      //array2 leading edge

                                if ((old&(1L<<k)) && !(onebuf[i]&(1L<<k))) {
                           if ((edge_status[j]&T0)&&(edge_status[j]&T1))//spike
                                 {
                             edge_status[j] &=~(T2|T3);
                                 }
                                 else {
                                      if ((edge_status[j]&T0) && edge[j]==Presence)
                                      {
                                             e[j] = 1;                      //second edge
                                             edge_status[j] |= T1;
                                      }
                                      else {
                                             e[j] = 0;                      //first edge
                                             edge_status[j] = T0;
                                             edge[j]=Trigger;
                                      }
                                      if (s->status & PLAY) t[j][e[j]]=((iFile-
1)*BUFFER_SIZE/2+i);
                                      else t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                 }
                                }

                               // array1 trailing edge

                                if (!(old&(1L<<j)) && (onebuf[i]&(1L<<j))) {
                                 if ((edge_status[j]&T0)&&(edge_status[j]&T1))
                                 {
                                      if (edge[j]==Presence) {
                                             e[j] = 2;                              //third
edge
                                             edge_status[j] |=T2;
                                      }
                                      else {
                                             e[j] = 3;                      //fourth edge
                                             edge_status[j] |=T3;
                                      }
                                      if (s->status & PLAY) t[j][e[j]]=((iFile-
1)*BUFFER_SIZE/2+i);
                                      else t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                 }
                                 else edge_status[j] &=~(T0|T1);           //spike
                                }

                                //array2 trailing edge

                                if (!(old&(1L<<k)) && (onebuf[i]&(1L<<k))) {
                                 if ((edge_status[j]&T0)&&(edge_status[j]&T1)) {
                                      if (edge[j]==Presence) {  //fourth edge
                                             e[j] = 3;
                                             edge_status[j] |=T3;
                                      }
```

```
                                        else {                                  //third edge
                                               e[j] = 2;
                                               edge_status[j] |=T2;
                                        }
                                        if (s->status & PLAY) t[j][e[j]]=((iFile-
1)*BUFFER_SIZE/2+i);
                                        else t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                  }
                                  else edge_status[j] &=~(T0|T1);    //spike
                                 }

                                        //calculate velocity, acceleration, length

                                 if (edge_status[j]==(T0|T1|T2|T3)){

 //front velocity
                                        delta_t = (t[j][1]-t[j][0])/(3600*1e4);
                                        if (delta_t<0) delta_t =
delta_t+qsize*BUFFER_SIZE/2/(3600*1e4);
                                        if (delta_t == 0) v[j][0] = 0;
                                        else v[j][0] = DISTANCE/delta_t;

 //back velocity
                                        delta_t = (t[j][3]-t[j][2])/(3600*1e4);
                                        if (delta_t<0) delta_t =
delta_t+qsize*BUFFER_SIZE/2/(3600*1e4);
                                        if (delta_t == 0) v[j][1] = 0;
                                        else v[j][1] = DISTANCE/delta_t;

 //acceleration
                                        delta_t = (t[j][3]-t[j][1])/(3600*1e4);
                                        if (delta_t<0) delta_t =
delta_t+qsize*BUFFER_SIZE/2/(3600*1e4);
                                        /*delta_v = v[j][1]-v[j][0];
                                        if (delta_t == 0) a[j] = 0;
                                        else a[j] = delta_v/delta_t;*/

 //length
                                        //l[j] = (v[j][0]*delta_t +
0.5*a[j]*delta_t*delta_t)*1609;
                                        l[j] = (0.5*(v[j][0]+v[j][1])*delta_t)*1609;

                                        edge_status[j] = 0x0;
                                        InvalidateRgn (hwnd, hrgnd, TRUE) ;
                                  }
                                 }
                                 old = onebuf[i];
                                }
#ifdef METER
                                for (i=0; i<BUFFER_SIZE/2; i++) {
                                 for (j=0, k=N_PAIR; j<N_PAIR && k<N_DIODE; j++, k++) {

                                        //array1 leading edge

                                  if ((old&(1L<<j)) && !(onebuf[i]&(1L<<j))) {
                                   if ((edge_status[j]&T0)&&(edge_status[j]&T1))    //spike
                                   {
                                        edge_status[j] &=~(T2|T3);
                                   }
                                   else {
                                        if ((edge_status[j]&T0) && edge[j]==Trigger)
                                        {
                                               e[j] = 1;                        //second edge
                                               edge_status[j] |= T1;
                                        }
                                        else
                                        {
                                               e[j] = 0;                        //first edge
                                               edge_status[j] = T0;
                                               edge[j]=Presence;
                                        }
```

```
                                        t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                    }
                                   }

                                        //array2 leading edge

                                   if ((old&(1L<<k)) && !(onebuf[i]&(1L<<k))) {
                               if ((edge_status[j]&T0)&&(edge_status[j]&T1))//spike
                                    {
                                 edge_status[j] &=~(T2|T3);
                                    }
                                    else {
                                        if ((edge_status[j]&T0) && edge[j]==Presence)
                                        {
                                               e[j] = 1;                    //second edge
                                               edge_status[j] |= T1;
                                        }
                                        else {
                                               e[j] = 0;                    //first edge
                                               edge_status[j] = T0;
                                               edge[j]=Trigger;
                                        }
                                        t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                    }
                                   }

                                  // array1 trailing edge

                                   if (!(old&(1L<<j)) && (onebuf[i]&(1L<<j))) {
                                    if ((edge_status[j]&T0)&&(edge_status[j]&T1))
                                    {
                                        if (edge[j]==Presence) {
                                               e[j] = 2;                           //third
edge
                                               edge_status[j] |=T2;
                                        }
                                        else {
                                               e[j] = 3;                    //fourth edge
                                               edge_status[j] |=T3;
                                        }
                                        t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                    }
                                    else edge_status[j] &=~(T0|T1);           //spike
                                   }

                                   //array2 trailing edge

                                   if (!(old&(1L<<k)) && (onebuf[i]&(1L<<k))) {
                                    if ((edge_status[j]&T0)&&(edge_status[j]&T1)) {
                                        if (edge[j]==Presence) {  //fourth edge
                                               e[j] = 3;
                                               edge_status[j] |=T3;
                                        }
                                        else {                                 //third edge
                                               e[j] = 2;
                                               edge_status[j] |=T2;
                                        }
                                               t[j][e[j]]=(qread*BUFFER_SIZE/2+i);
                                    }
                                    else edge_status[j] &=~(T0|T1);    //spike
                                   }

                                               //calculate velocity, acceleration, length

                                   if (edge_status[j]==(T0|T1|T2|T3)){

 //front velocity
                                        delta_t = (t[j][1]-t[j][0])/1e4;
                                        if (delta_t<0) delta_t =
delta_t+qsize*BUFFER_SIZE/2/1e4;
                                        if (delta_t == 0) v[j][0] = 0;
```

```
                                        else v[j][0] = 0.2/delta_t;

 //back velocity
                                        delta_t = (t[j][3]-t[j][2])/1e4;
                                        if (delta_t<0) delta_t =
delta_t+qsize*BUFFER_SIZE/2/1e4;
                                        if (delta_t == 0) v[j][1] = 0;
                                        v[j][1] = 0.2/delta_t;

 //acceleration
                                        delta_t = (t[j][3]-t[j][1])/1e4;
                                        if (delta_t<0) delta_t =
delta_t+qsize*BUFFER_SIZE/2/1e4;
                                        delta_v = v[j][1]-v[j][0];
                                        if (delta_t == 0) a[j] = 0;
                                        else a[j] = delta_v/delta_t;

 //length
                                        l[j] = v[j][0]*delta_t + 0.5*a[j]*delta_t*delta_t;

                                        edge_status[j] = 0x0;
                                        InvalidateRgn (hwnd, hrgnd, TRUE) ;
                                    }
                                   }
                                   old = onebuf[i];
                                  }
#endif

                                  qread++;
                                  qread %= qsize;
                           }

                           if(!(s->status & PLAY_EOF))
                           {
                             if (exmove)
                               ScrollWindow (hwnd, -1, 0, &RePaint, NULL);
                             else
                               ScrollWindow (hwnd, -DX, 0, &RePaint, NULL);
                           }
                    }
                   return 0 ;

          case WM_DESTROY :
                           PostQuitMessage (0) ;
                           //MFree();
                           //sensorClose( s );
                           DeleteObject (  hrgnu );
                   return 0 ;
     }
     return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
 //HWND  hwndParent;
 //HDC   hdcParent;
    //hwndParent = GetParent (hDlg);
     switch (iMsg)
          {
          case WM_INITDIALOG :
               return TRUE ;

          case WM_COMMAND :

               switch (LOWORD (wParam))
               {
                           case IDD_RECORD :
                                   Record();
                                   return 0;
                           case IDD_PLAY :
                                   Play();
```

```
                                        return 0;
                                case IDD_STOP :
                                        RPStop();
                                        return 0;
                                case IDD_PAUSE :
                                        FilePause( s );
                                        return 0;
                                case IDD_CONT :
                                        FileCont( s );
                                        return 0;
                                case IDD_INPUTP :
                                case IDD_INPUTR :
                                        GetFileName();
                                        return 0;
                                case IDD_END :
                        case IDCANCEL :
                             EndDialog (hDlg, 0) ;
                                 return TRUE ;

                }
                break ;
            case WM_CLOSE :
                    DestroyWindow (hDlg);
                    hDlgModeless=0;
                    KillTimer (hwnd, 1) ;
                    FileClose( s );
                    sensorClose( s );
                    break;
            }
    return FALSE ;
}
```