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

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,
Zhaoqing Wang, Bo Chen
University of California, Davis

California PATH Research Report
UCB-ITS-PRR-2002-23

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.

Final Report for TO 4116

July 2002

ISSN 1055-1425

CALIFORNIA PARTNERS FOR ADVANCED TRANSIT AND HIGHWAYS

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
Zhaoqing Wang
Bo Chen

Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California, Davis
Davis, CA 95616

April 8, 2002

Contents

1. Introduction

Travel time is the most important aspect of the Intelligent Transportation System (ITS). According to the conclusion which Both CalTrans and the US Department of Transportation have come to , 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 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). While we have previously developed a real-time laser-based non-intrusive detection system for measurement of true travel time on the highway, we improved the system further. A feed-back loop, using microchip control, was introduced into the system. This makes the system easy to use. The mechanical components were also improved to increase the precision of the system. Software to display the information or communicate with other research groups was updated. The new system device is shown in Figure 1-1 .

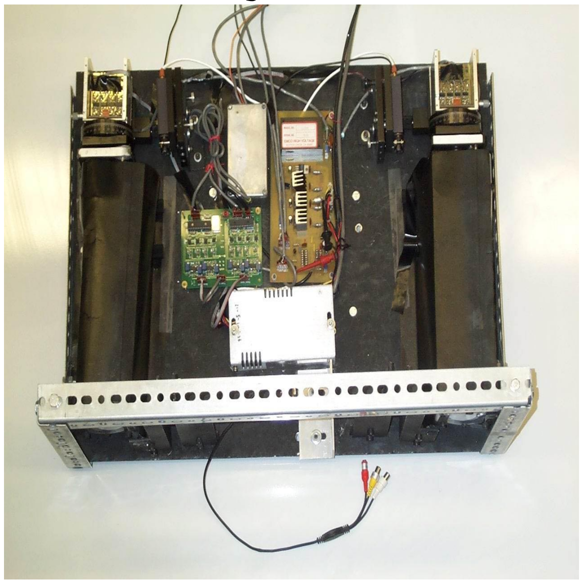


Figure 1-1 new system device

2. Overview of the System

Figure 2-1 shows the entire traffic detection system schematic. It is composed of laser units, optics, photodiode sensors, electronic circuits and a microprocessor, an optical isolator used for

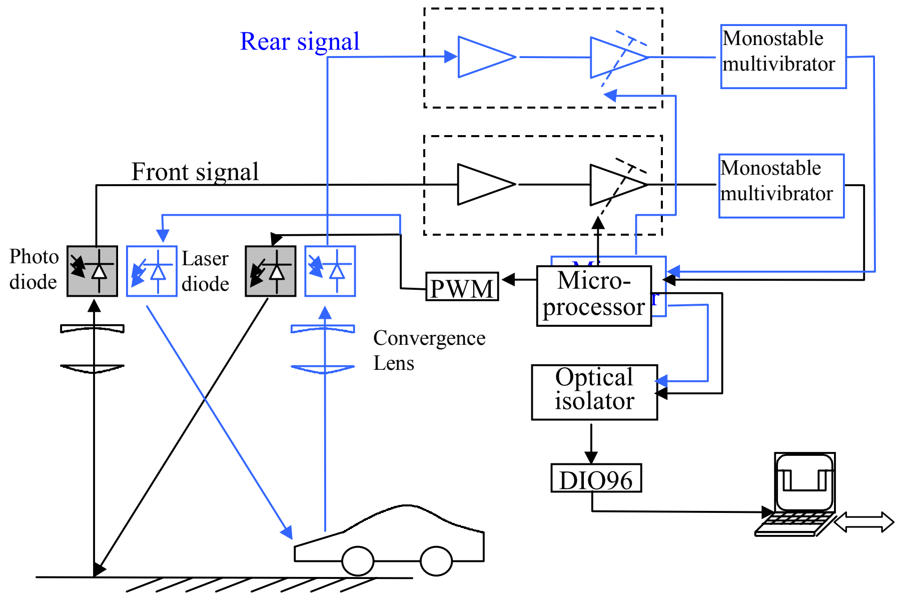


Figure 2-1 laser-based detection system schematic diagram

communicating with a computer, mechanics for adjusting the laser beam and sensor position, and a computer to implement the software and provide a user interface. All components are mounted on a back-board except the computer. The traffic detection system operates as follows. The system is mounted above a roadway. Two laser units project two laser beams to the roadway. The reflected light from each laser beam is detected by an associated sensor. Each sensor converts the laser signal into an electronic signal, which is then processed by electric circuits and a microprocessor. Finally the electric signal is transmitted to the computer via the optical isolator. Each system component will be described in detail hereinafter.

2.1. *Integrated Laser and Optics*

Two off-the-shelf integrated high power diode laser systems, from Power Technology Inc (model ML20A15-L2) are used in the current prototype. This is an integrated laser system that incorporates a DC/DC voltage converter, a voltage regulator, a pulse generator, a laser diode, and line generation optics all into one unit. Operation only requires 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 up to a maximum rate of 10 kHz. The line generating optics produce a beam having a full fan angle of 15 degrees. The laser's high performance and compact size make it a good candidate for use in the field deployable prototype system.

A laser producing infrared light (with 905nm wavelength) was chosen for a number of reasons. Infrared light has good transmittance through fog, facilitating better performance under a larger range of weather conditions. Furthermore, the intensity of sunlight near the wavelength of the laser is a local minimum, minimizing 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, causing no distraction to passing motorists.

The laser has to be properly adjusted to obtain a fine line. The procedure for adjusting is as follows:

a. Adjust the focus without the cylindrical lens in the line generator.
b. Put the cylindrical lens on and rotate it to minimize the width of the line.

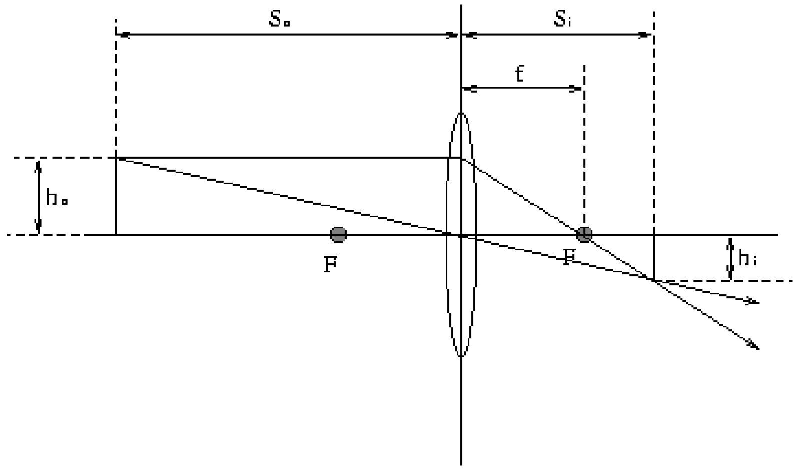


Figure 2-2: Telescopic Lens Parameter

In order to retain focus, a clamp for the cable connector is used to hold the setting of the laser module line generator.
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 several criteria. It should have an adjustable focal length within a range around the desired focal length, a field-of-view large enough to capture the width of an entire lane, and 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), the unit will be mounted about 6.40 m (21.0 ft) above the roadway (s_o), and 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. It was determined that s_i = 15.8 mm (0.620 in).

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

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 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 2-2 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 width of the uncorrected field-of-view of the sensor to the desired field-of view. The desired field-of-view X, is determined from 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. To insure reliable vehicle detection it is important that *Hc* be below the bumper height of most common vehicles.

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

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. 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*). This causes objects to the left to appear 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.

2.2. Electronics

The system electronics includes 5 layers, shown in Figure 2-3:

1. Signal Processing: amplifies analog signals and filters noise, converts analog signals to digital signals
2. Device Driver: transmits signals from the circuit board to the computer. Defines how the interface operates.

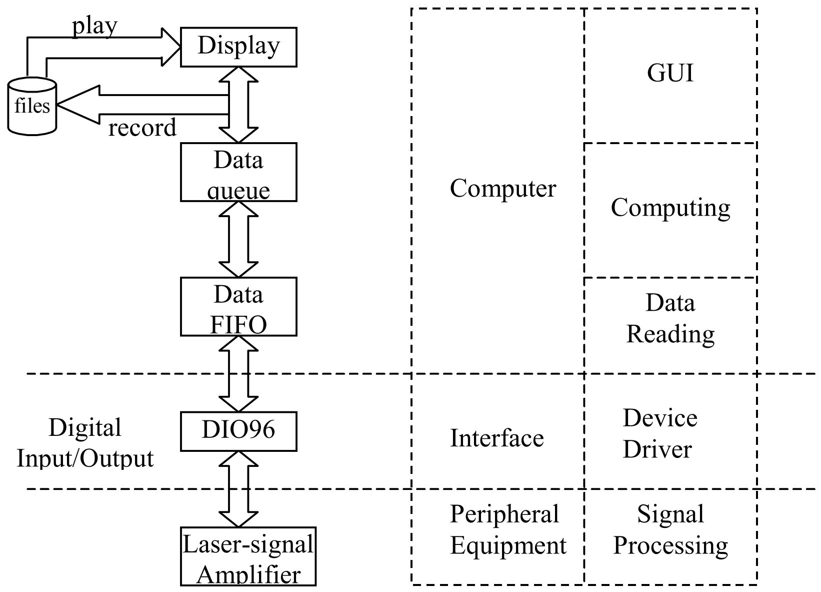


Figure 2-3 system architecture

3. Data Reading: reads signals from the interface and stores data in RAM or onto the hard disk
4. Computing: calculates speeds, lengths, and accelerations of vehicles from acquired data.
5. GUI: displays calculation results in a user-friendly way.

According to the principle of detection, the system needs only to distinguish vehicle presence under the laser lines. It is advantageous 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, and therefore reduce the computer system requirements. The implementation of this method is based on the high signal-to-noise ratio of the new circuitry. The hardware consists of five parts: the power supply, the laser components, the amplifiers, a microprocessor, and the digital I/O board and computer.

The high voltage pulse generator for the laser diode is built into the laser unit, 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 well. 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 system cost. 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 circuit. 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 circuitry time response, so this method will improve the reliability of the system time response and allow us to gather more channels of the signal (i.e.24 channels). Using a digital signal as output from the hardware will significantly simplify the signal processing in the software. The implementation of digital output 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. Thus the cost of new circuitry is only one-fifth of that previous cost. This is vital 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 2-4`.

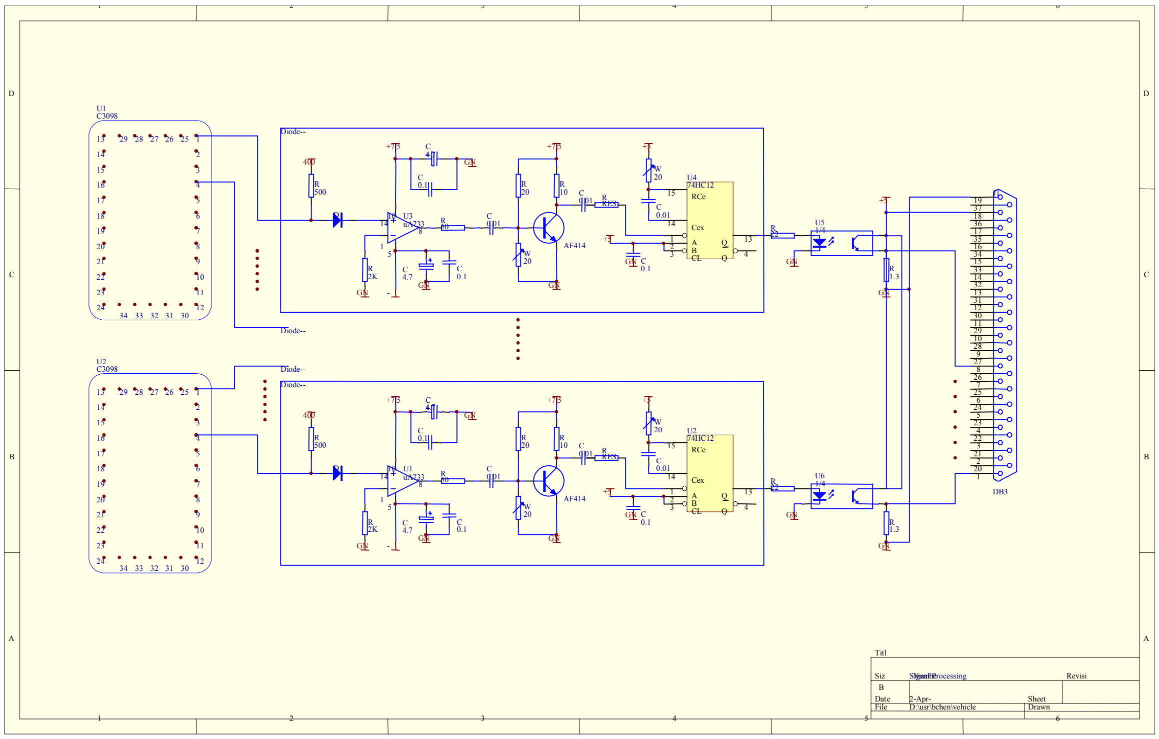


Figure 2-4: Circuitry of Sensor Electronics

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 the amplifier and generates high level output when there is no vehicle under the system. When a moving vehicle blocks the laser, no pulse is present to trigger U2, and the output will be low. Capacitors are used to isolate DC signals between different parts of the circuit, so that a continuous signal can not pass through the circuit. Ambient lights normally generate continuous signals. After intensive testing and adjustment, the interference between different parts of the circuits are reduced to the lowest level, so that the system has a signal/noise ratio 5 times better than the previous version. The transition time is an important factor to the accuracy and consistency of data measured.

The potentiometers (W1) are used to adjust the sensitivity of the analog circuit output. Because

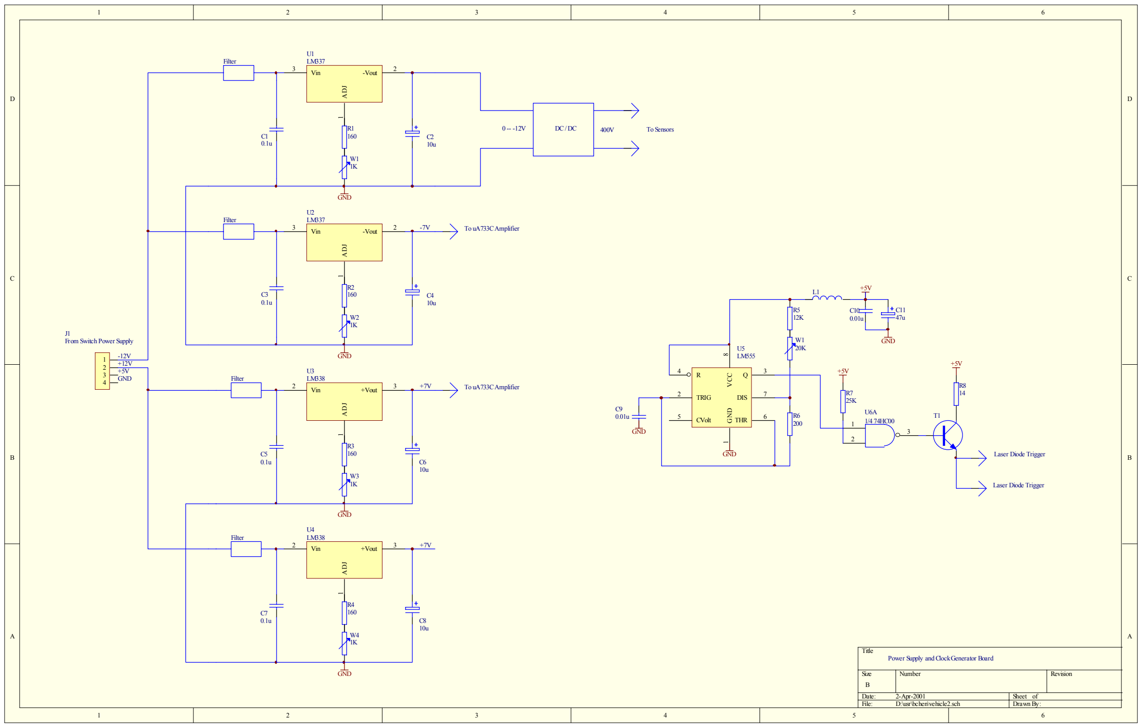


Figure 2-5: Schematic of power supply.

there are eight signal channels in the current system, manual adjustment of the potentiometers takes a long time at the test site. It will be more difficult if the system is extended to 48 signal channels. For convenient adjustment, a microprocessor was introduced into the circuitry to automatically adjust the potentiometers. Using this microprocessor, the adjustment can be done in less than one second. We also expect the accuracy of the adjustment to be much higher than manual adjustment.

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.

Adjustment procedures of electronic circuitry:
1. Adjust the angle of the laser to obtain the maximum signal.
2. Attach the oscilloscope probe to pin 7 of ua733 to measure the signal amplitude.
3. Using microprocessor adjust potentiometer to the proper threshold.

2.3. Mechanical

`Figure 2-6` shows the configuration of the newly designed system. The new mechanical design of the optics is flexible and allows the optical components to be adjusted to optimize alignment and focus of the laser beam The cylindrical lenses, image lenses, and APD sensors are mounted on optical rails, which are mounted to the base plate. A holder that is adjustable in two dimensions clamps each optical component and sensor. 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 rotating lens holder. As a result the sensor can be adjusted via both translation and rotation.

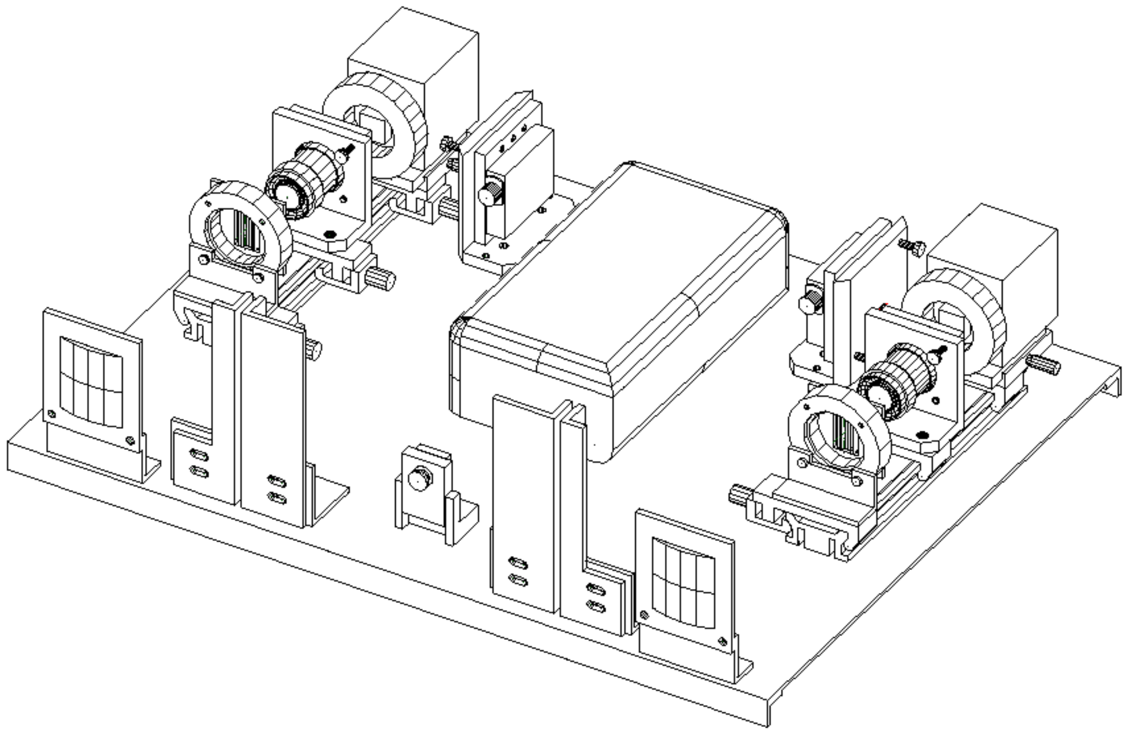


Figure 2-6 Mechanical Configuration of the Detection System.

Because the new optical system is flexible, it is necessary to define a procedure to adjust the system. In the optical system shown in `Figure 2-7`, the adjustment includes optimizing the cylindrical telescope, and focusing and aligning optical components and lasers. We have developed a procedure for adjusting the optics of the system. :

* Align the 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.

A clear image of the reflected laser is obtained by following the defined procedure. The optimization of the optics significantly improves (i.e. lowers) the critical height and eliminates unexpected laser reflection from reflective vehicle surfaces. A visible laser and a CCD camera are used to verify the image quality of the optical system.

2.4 Operating System Platform and User Interface

Data acquisition software is used to collect, process, and display data from the detection system. Periodically data has to be read from the electronic sensor output. All tasks must be done simultaneously. However, the data has to be acquired in the same interval from the hardware. In other words, the data acquisition task should be deterministic, otherwise the information will be lost. The concurrence of the deterministic tasks requires real-time performance of the operating system.

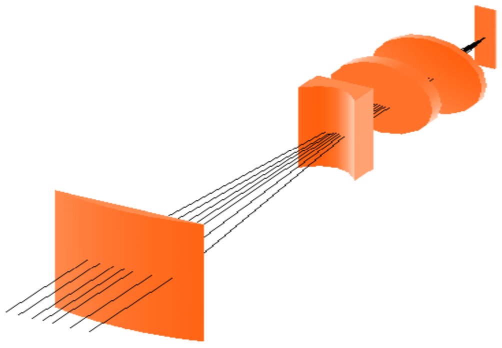


Figure 2-7: Optical System

We chose a real-time Linux platform for our new system. Pt-Linux is becoming increasingly popular in real-time embedded systems. it is one of various real-time systems and was developed by Victor Yodaiken and Michael Barabanov from the Department of Computer Science at the Institute for Mining and Technology of New Mexico. RT-Linux builds a small kernel directly over the processor, which is independent of the Linux kernel. The Linux kernel runs on top of this kernel and shares the processor with other RT-tasks. Because of its small kernel, RT-Linux has exceptional real-time performance, as long as real-time tasks are not overly complex. In our case, the most critical task is to read data from I/O ports in an accurate interval. This task is suitable for RT-Linux. In RT-Linux, device drivers are kernel modules that always run before user space processes. While data acquisition is executed in kernel space in real-time mode, the data processing and GUI program can run in the regular Linux OS.Since the outputs of the sensor circuits are digital signals now, 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 as an input interface between the sensor circuitry and the computer system. This digital I/O has 96 channels of configurable input/output and will be suitable for our system when all 48 channels of signal are required.

3. The Problems of the Old system

3.1. Platform

The old data acquisition system uses a PMAC DSP board and A/D converter as the data input device. The data acquisition software is a multi-thread program running in a real-time Lynx operating system. The data from the laser detection system is acquired by the PMAC board in a certain interval and is written into a dual-ported memory buffer by the device driver. The user level application software reads data from the device driver buffer and processes it in different threads to calculate vehicle data and display results in a GUI. The most critical real-time tasks are handled primarily by the PMAC board. Because the implementation of data acquisition using a DSP board makes the system expensive, we have been seeking a real-time system to conduct both data acquisition and processing.

3.2. No-feedback : It takes a long time to manually adjust the system to function properly.

Potentiometers are used to adjust the sensitivity of the analog circuit output. The adjustment must be fine in order to get the proper gain. If it is too large the output level will be high even if the laser beam is blocked if it is too small the laser signal can not be detected by the system. Because there are eight signal channels in the current system, manual adjustment of the potentiometers takes a long time at the test site. Therefore Adjustments will become increasingly difficult if the system is expanded to 48 signal channels. On-site manual adjustment is also difficult because the gain precision can not be assured.

3.3. Uneven Sensor Distribution

The final outputs of the detection system are speeds, lengths, and accelerations of vehicles. An algorithm to calculate them is introduced. It is based on the uniform distribution of the sensor elements. However the distribution of the ADP array is uneven causing the results gathered from the computer to vary slightly from the real values of the vehicle This should be compensated for in the software.

To solve the problems mentioned above, we changed the software platform to a real-time Linux operating system in the new version. Furthermore, a microchip was introduced to adjust potentiometers automatically. This microchip is also used to filter the noise in the circuits. According to position analysis of the APD array, the algorithm to calculate vehicle speeds is compensated for in the software. The improvement of each problem will be described in detail in the following chapters.

4. Real-Time Device Driver and Data Acquisition Software

From a user's point of view, the result should be easily readable and recognized. Thus a GUI (graphical user interface) was introduced in the system. The system uses Xwindow (Unix program) to display the results. The software is based on the Linux operating system. The open software Linux can be installed in a workstation , a PC, or other machines. The input/output interface converts the signals into data that the computer can process. A PCI-DIO96 board is used as the I/O device.

4.1. Device Driver

A PCI-DIO-96 general-purpose digital I/O board from National Instruments is used in the new data acquisition system. This board has a 96-channel programmable digital I/O, which will be suitable for when our system is extended to 48 channels.
The PCI-DIO-96 board uses the PCI MITE ASIC to communicate with the PCI bus. The base address and interrupt level for the board are stored inside the PCI MITE at power on. In order to configure the PCI MITE chip, the function call *pci_read_config_byte()* is used to search the PCI configuration space for the National Instruments vendor ID and PCI-DIO-96 device ID. If the board is found, the function call stores all the board's configuration information into a data structure *pci_dev*. Base Address Register 0 (BAR0) corresponds to the PCI MITE base address, while Base Address Register 1 (BAR1) is the base address of the board register.
To configure the board the value *0x0000eaea* must be written to offset *0x340* from the PCI MITE address.
The PCI-DIO-96 device driver searchs the board in the PCI bus and configures the register in the board. The I/O port of the PCI-DIO-96 is memory mapped. The base address can be mapped by the *ioremap()* function to kernel memory space.
FIFOs are used to communicate and transfer data from real-time kernel and user space. Two FIFOs are created as buffers. The data is written to data FIFOs in turn in kernel space. When one FIFO is full of data, the device driver will notify the process in user space that the data is available through another FIFO. The third FIFO is used to synchronize the data written and read between kernel space and user space. The forth FIFO is created to send commands from the user space to the kernel space. These commands will set the sampling rate and the start and stop of the sampling thread in the device driver. In user space the data read is synchronized using the *select* function call. Each of the device driver's routines for the PCI-DIO-96 device are described below and the device driver code is listed in the Appendix.

unsigned long dio96_setup(void)

Search the PCI-DIO-96 board in the PCI bus, configures the board, and maps the base address to the kernel memory space.

```
int init_module(void)
```

Initialize the device driver kernel module. It creates FIFOs for data communication and synchronization, threads to read data from I/O ports, and register FIFO handlers.

```
void *thread_code(void *t)
```

Thread code to read data from I/O ports and synchronize the data read/write to and from FIFO between kernel and user spaces.

```
int msg_handler(unsigned int fifo)
```

The FIFO handler that the application program sends commands to device driver to configure board I/O port, set the sampling period, and start and stop sampling. When the application program writes to this FIFO, the handler reads the data, writes a command to the register of the PCI-DIO-96 board, and wakes up the periodic thread. If the STOP command is sent, the handler will suspend the periodic thread.

```
int data_handler(unsigned int fifo)
```

The handler of a FIFO to which the application program sends a signal to notify that the data in the specific data FIFO has been read. If the buffer is not read before the device driver writes new data to that data FIFO, an error message will printed out to indicate the overrun.

4.2. Real-Time Data Acquisition Software

The Lynx OS on which the old software is based, is a hard real-time operating system. How long each process runs is determined by one of the scheduling policies selected (Default, Round Robin, and FIFO). For the default scheduling, a "time quantum" value is established by the constant QUANTUM, defined in param.h. The time quantum is used to determine how long each process at a particular priority level will run before giving up the CUP to the next process (at that level). As long as there are processes that are ready to run in the queue for that priority level, the scheduler will schedule them to run in a round-robin fashion, with each process running for that priority level's established quantum time.

LinuxThreads provides kernel-level threads created with the new *clone()* system call, and all scheduling is done in the kernel. Therefore LinuxThreads is an implementation of soft real-time. In Linux the process priority values have a range from -20 to 20, while in the Lynx OS process priority values range from 1-255. When porting code from Lynx to Linux, the priority values should be changed. By changing the priority values and queue sizes to fit those in Linux , all tasks can be fulfilled before the deadlines. This ensures that no data is lost.

Both the Lynx OS and LinuxThreads implement Posix 1003.1c. However the Lynx thread is based on the interfaces defined by Posix Draft 4, while LinuxThread is based on the final standard. The Posix Draft 4 interfaces vary significantly from the final standard Posix 1003. The differences are described below.

In the Lynx OS, the *pthread_set_prio* function is used to set a thread's scheduling priority attribute. In LinuxThread the *pthread_attr_setschedparam* function is used to set the priority. Listing 1 is a segment of code used to initialize and setup a thread attribute object and to create a thread in Linux. Structure `sched_param` contains a single member that specifies the scheduling priority.

```
pthread_attr_t read_attr;              /* thread attributes */
struct sched_param  read_param;        /* used by pthread_attr_setschedparam() */
.
.
.
```

```
  if (pthread_attr_init(&read_attr)<0) /* create thread attributes */
    perror("sensorGatherArray(): pthread_attr_create()"), exit(1);

  pthread_attr_setinheritsched(&read_attr, PTHREAD_EXPLICIT_SCHED);
  pthread_attr_setschedpolicy(&read_attr, SCHED_RR);
  pthread_attr_setdetachstate(&read_attr, PTHREAD_CREATE_DETACHED);

  read_param.sched_priority = getpriority(PRIO_PROCESS,0)+PRIO_READ;

  pthread_attr_setschedparam(&read_attr, &read_param);
  pthread_attr_setschedparam(&record_attr, &record_param);

  if (pthread_create(&read_tid_sensor, &read_attr, sensorThread, s)<0)
    perror("sensorGatherArray():pthread_create(read_tid)"),exit(1);
```
.
Counting semaphores are used to synchronize data read and written from queues. Lynx OS actually provides two types of counting semaphores, System V-compatible and Lynx proprietary. The later is used in our software. The following table shows the difference of the features and syntax of the counting semaphores in Lynx and Linux.

Features	Syntax	
	Lynx	Linux
Create counting semaphore	csem_create_val	sem_init
Suspend thread	csem_wait	sem_wait
Suspend thread-no blocking	csem_wait	sem_trywait
Signal semaphore	csem_signal	sem_post

Even though LinuxThread is soft real-time, the tasks which are not time-critical can still be implemented as long as the data buffer sizes are large enough. We set the buffer size twice as in Lynx and there is no observed data lost.

Some new functionality has been added into the new software in Linux system. In the old software, the real-time signals are displayed continuously, so it is hard to see the detail of signals when vehicles passing under the detection system. In the new software, a one-shot mode is used to capture the signals so that the signals will stay on the screen (until the display is reset) after a vehicle has passed the system.

5. Microprocessor Closed-loop Control Unit

A microprocessor was introduced into new system to adjust the potentiometers automatically and implement signal processing. The microprocessor we have chosen is a PIC 16F876 from Microchip, Inc. Microchip's RISC-based MCUs are designed for applications requiring high performance and low cost. MCUs combine high performance, low cost, and small package size, offering the best price/performance ratio in the industry. The PIC16F87X Microprocessor(MCU) family provides a migration path from OTP to FLASH in 28 to 44-pin packages, with a wide range of peripheral integration options. This family features a 14-bit instruction set, 5 to 8 channels of 10-bit analog-to-digital converters, interrupt handling capability, various serial interface capabilities, Capture/Compare/PWM, brown-out detection, and an 8-level deep stack. The PIC16F87X family provides performance and versatility to meet the most demanding requirements of analog designs. With FLASH program memory, PIC16F87X devices can also be reprogrammed over the entire operating voltage range. PIC16F876 has up to 8K*14 words of flash program memory, 378*8 bytes of data memory (RAM), and 256*8 bytes of EEPROM data memory. It can run at clock speed of 20 MHz. This Microprocessor meets the requirements of our system at a reasonable cost.

5.1. The Principle of Closed-loop Adjustment

The system uses a closed loop to control the amplifier. The schemetic of the feedback controller is shown in `Figure 5-1`. Here (a) is the first amplifier and **β** is the second amplifier. And feedback is through (b). The system output is connected to F. Assume the critical input value of trigger (0->1) is $I_{c,.}$ The critical value of noise , which can not trigger the D from 0 to 1, is L_n. The critical value of signal reflected from the ground, which enable the D trigger from 0 to 1, is L_s. We can get the formulation as follows.

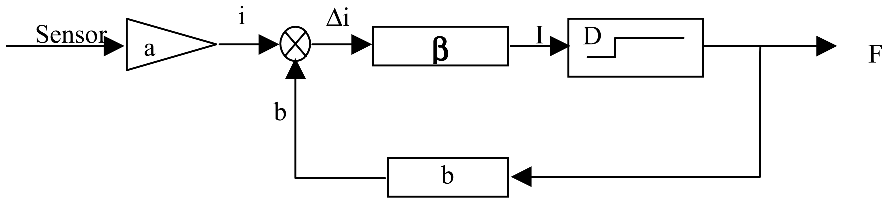


Figure 5-1 **feed back diagram**

$$\Delta i = i - b$$

$$I = \Delta i * \beta$$

$$F = \begin{cases} 0\text{: If the laser beam is blocked by vehicle} \quad \text{-> } I < I_c \\ 1\text{: If the sensor can receive the reflected laser beam from the ground -> } I > I_c \end{cases}$$

If F = 1, then

$$(i-b)*\beta > I_c$$

If $b < i - \frac{I_c}{\beta}$, the output F will be the high.

If $i = 0$, the output F must be : $b*\beta > -I_c$

We must set a certain limitation I_n, if $I < I_n$ so that F is not triggered by noise, so the output should still be 0.

$$b > I_n - \frac{I_c}{\beta}$$

if we want to pick up the signal of laser beam reflected from the ground, for example, when $I > I_s$, the output F must be 1.

$$b < I_s - \frac{I_c}{\beta}$$

so, $$b \in [I_n - {}^{I_c}/_{\beta}, I_s - {}^{I_c}/_{\beta}]$$

From the equation above, we can get the conclusion that the bigger the difference between signal of the laser and noise, the wider the range of b. It means that the system will be more reliable and more steady.

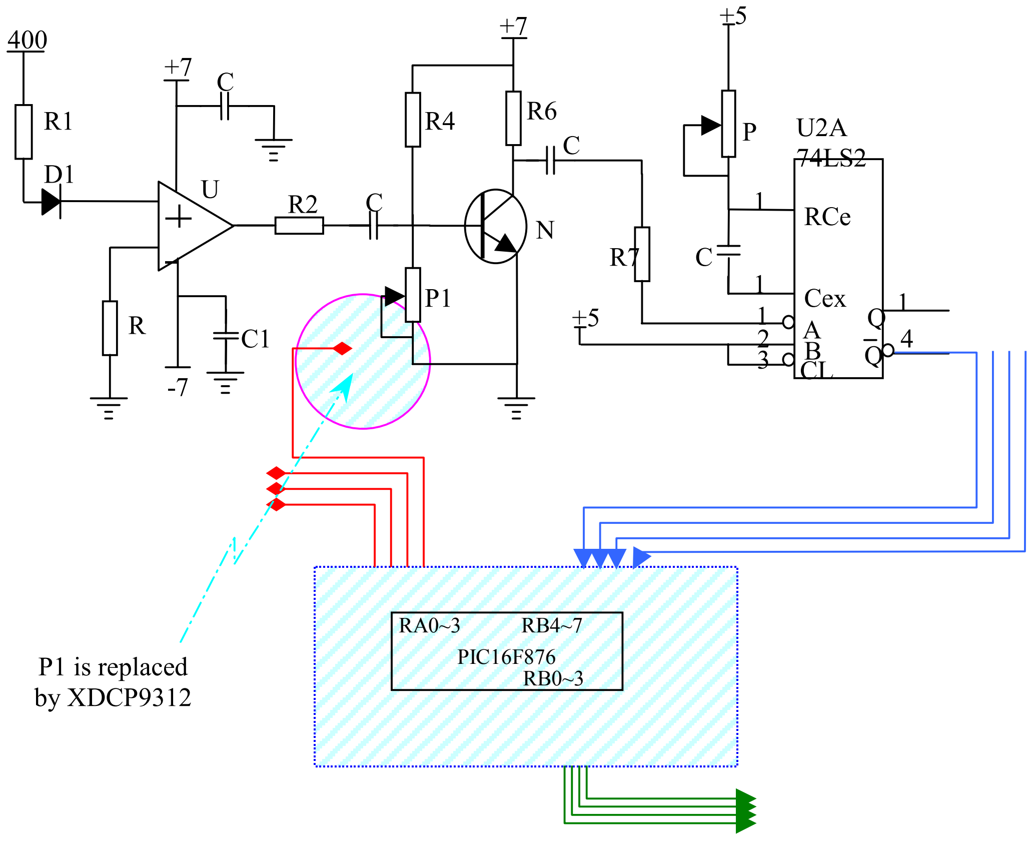


Figure 5-2 The Principle of Closed-loop Adjustment

Figure 5-2 shows microprocessor controlled digital potentiometer adjustment principle integrated into the original signal amplification and processing circuitry. The potentiometer P1 is replaced by a digital potentiometer XDCP-X9312. It is a 100-step digital potentiometer controlled by digital pulses. The digital output of 74HC123 is fed back into RB4-7 of port B in the microprocessor to adjust the potentiometer. The output signals are sampled and time-stamped. When the system is triggered to perform the automatic adjustment, the potentiometers are set to zero. The microprocessor then sends a control pulse to the digital potentiometer to increase the value of the resistor. Each microprocessor controls 4 sensor channels, The hardware and software of microprocessor adjustment unit is discussed in detail below.

5.2. The Closed-loop Control Unit Hardware

The detailed closed-loop control unit circuit is shown in Figure 5-3. The signals from 74HC123

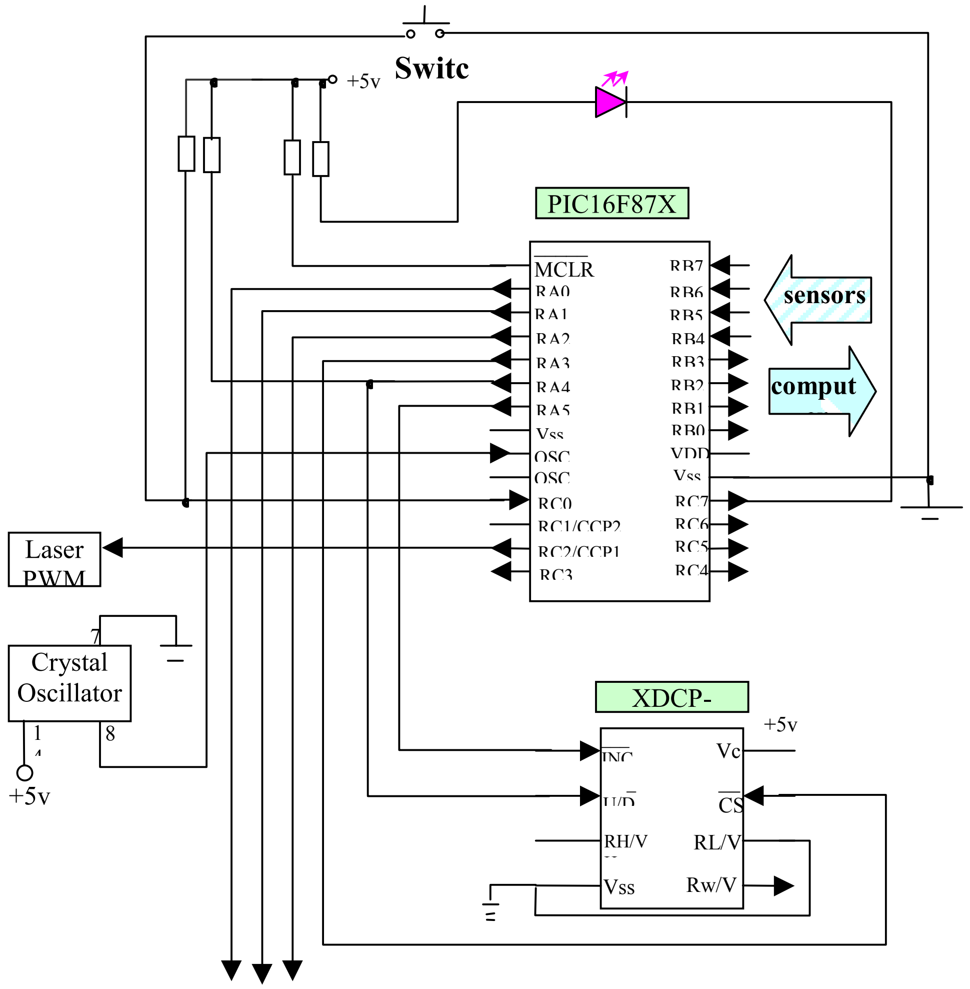


Figure 5-3 Microchip Control unit circuits

are introduced into RB4-RB7 of microprocessor. The microprocessor processes the signals and send output pulses to the digital potentiometers to adjust their values. A switch is used to initiate automatic adjustment. A LED flashes whenever the adjustment is in progress.

$\overline{\text{RA0}}$~RA3(output): cs of XDCP for channel 0~channel 3,

RA4 (output): $\overline{\text{INC}}$ of XDCP for all channels ↘

RA5(output): U/$\overline{\text{D}}$ of XDCP for all channels. If 1, the potentiometer value is increased, otherwise it is decreased.

OSC1: CLKIN ;
OSC2: CLKOUT ;
RC0: If RC0 is set to 0, the system is in adjusting status, otherwise it is in work status filtering noise. It is inputted from the switcher.

RB4~RB7(input from sensors): feedback of channel0~channel3. A change of signal in the port causes an interrupt.
The XDCP's potentiometers resistors can be adjusted from 100 to 10 (k Ω)

The value of the digital potentiometer is increased or decreased by one step whenever a pulse is applied to the INC pin. In the microprocessor, RA5 of port A outputs pulses to control the value of the digital potentiometers according to the input data from the output of 74HC123. RA0~RA3 are connected to CS (chip select) of the digital potentiometer to select the chip that is currently adjusted. RA4 controls the potentiometer increase and decrease when there is a pulse presented on pin INC. The microprocessor clock is provided by a 20 MHz crystal oscillator that is connected to OSC1. The microprocessor also provides a trigger signal to the diode lasers, so there is no need to use a clock circuit for laser triggering.There are two microprocessor states, adjustment and working status. In the adjustment status, which is enabled by applying LOW to RC0, the microprocessor takes feedback from the output and adjusts the digital potentiometers to proper positions. After adjustment completion, RC0 is set to HIGH and the potentiometer values are maintained. The adjustment status is initiated by pushing a button switch, which causes an interrupt in the microprocessor. In working status, RC0 is set to HIGH and the microprocessor begins signal processing. The processing removes some unwanted signals from the vehicle reflection. The processing implementation is based on the detection of signal periods. The unwanted reflection signals are removed by determining the signal duration. If the signal duration is less than a specified amount, it is considered an unusual signal.

5.3. Microprocessor Software

Figure 5-4 shows the flowchart for adjusting the digital potentiometers. The software is written in C. In the beginning of adjustment, all potentiometers are set to zero. When the start button is pushed and an interrupt occurs, the program starts adjusting by increasing the digital potentiometer one step. It then reevaluates the signal to check if the potentiometer is in the correct position. This process continues until the values of all potentiometers are proper and the signals are normal. When we get proper values of Digital controlled potentiometers, Microchip give a signal to lock the value of XDCP, which can be kept after power on next time. And LED the indicator will be on one of three statuses:

a. Normal (ligh): a proper value gotten
b. Flash (slow): value may be equal to zero (the system should be adjusted)
c. Flash (fast): value may be equal to maximum 10k ohm (the system should be adjusted)

Considering of keep time of potentiometer, this piece of program have some delays of time. This progression needs some milliseconds to finish.

Figure 5-5 shows a signal processing flowchart. The signal processing starts at the interrupt caused by the input status change (from sensor electronics) of the microcontroller. The processing is also based on the detection of the signal periods. The number of interrupts is stored in a register. If this number is less than a preset value FILT_NUM, the signal causing these interrupts is considered an abnormal signal and will be taken out of the output. Also, if the signal has an incorrect period, the signal is considered noise and is deleted from the output. The main modules of microprocessor program are described below. The source code is listed in Appendix.

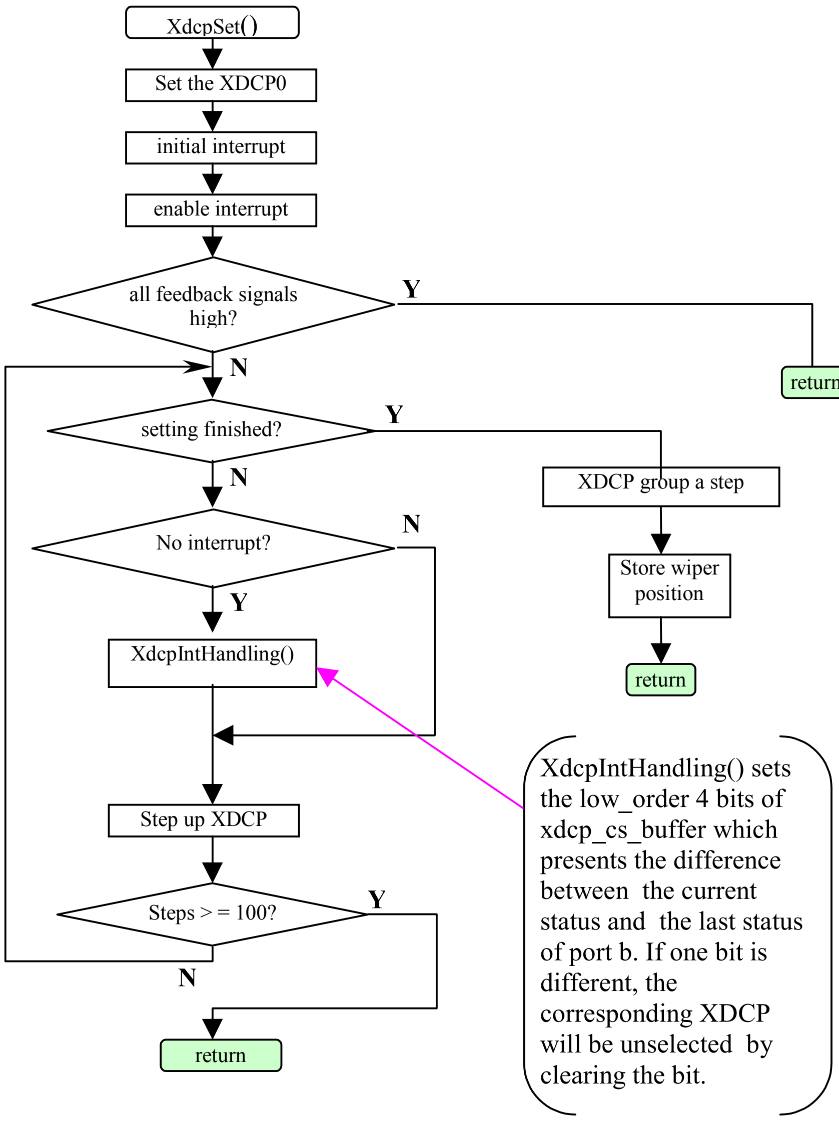

XdcpSet()
Set the XDCP0
initial interrupt
enable interrupt
all feedback signals high?
Y
return
N
setting finished?
Y
XDCP group a step
Store wiper position
return
N
No interrupt?
N
Y
XdcpIntHandling()
Step up XDCP
Steps > = 100?
Y
N
return
XdcpIntHandling() sets the low_order 4 bits of xdcp_cs_buffer which presents the difference between the current status and the last status of port b. If one bit is different, the corresponding XDCP will be unselected by clearing the bit.

Figure 5-4 Flowchart of XDCP Adjustment

`interrupt()`

The interrupt handler stores the 16-bit time stamp and port B status in an array of 24 bytes, when an interrupt from port B<4:7> is received. The time stamp is stored in the first two bytes and the status is stored in the third byte. Therefore the array can store the time stamps and statuses of 8 interrupts.

`XdcpSet()`

The XDCP is set to the proper value so that laser signal can be detected by the digital output circuit.

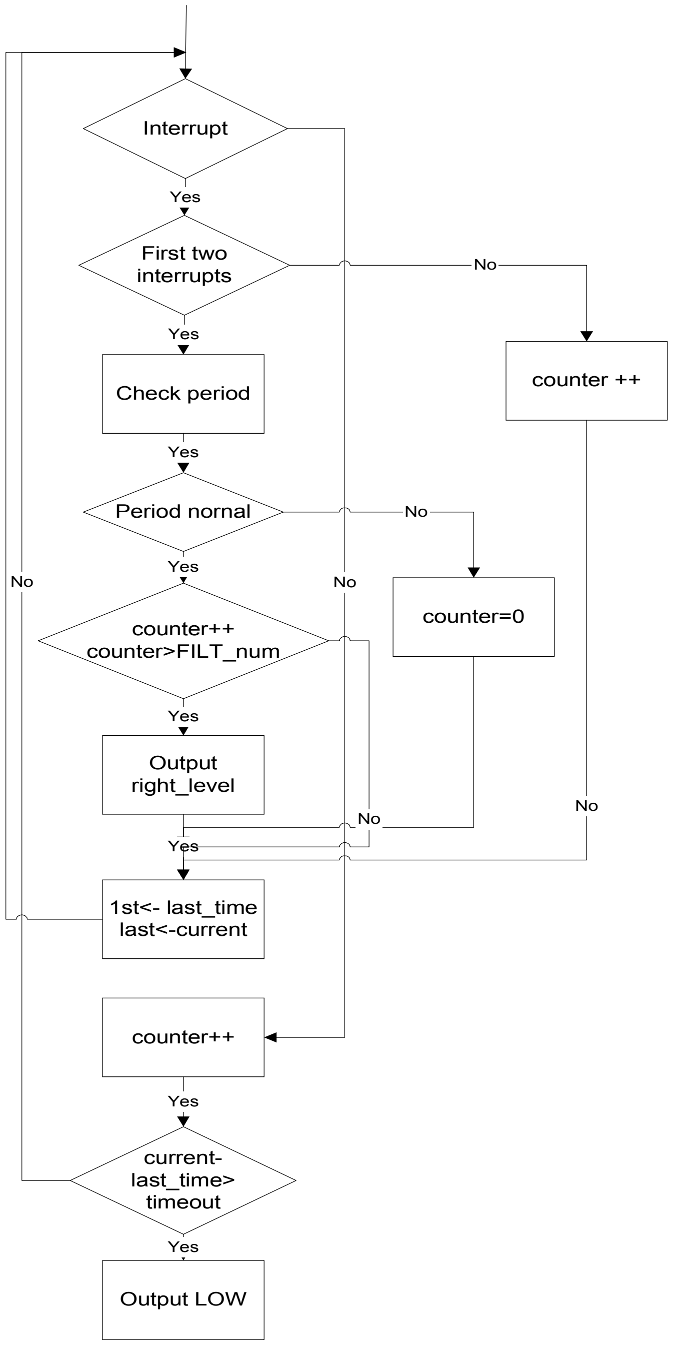


Figure 5-5 Flowchart of Microprocessor Signal Processing

`XdcpIntHandling()`

When the adjust button is pushed, an interrupt is generated and the interrupt counter is incremented. This function is called by XdcpSet() when the interrupt counter is not 0. This function also reads data from the interrupt status array, determines which channel(s) cause the interrupt, and sets the channel selection varible (xdcp_cs_buffer).

`FlashLed()`
Outputs a pulse to the port c pin 7, causing an LED to blink.

`Pwm()`
Set capture/compare/PWM Register1 for pulse width modulation (PWM). Refer to PIC16F87X manual (p57) for details.

`Init()`
This function initializes the 16f876 microchip.

`XdcpGoZero()`
This function initializes the values of all potentiometers to 0.

`XdcpGoUp()`
This function increments XDCP by one step. XDPC must be enabled and set to up before calling this function and disabled after calling this function.

`XdcpGoUpHalf()`
Adjust the potentiometer to half of its maximum value.

`DetectPeriod()`
This function detects the pulse period. The period is obtained by averaging 8 successive periods. One period is obtained from 4 interrupts.

`ReadTime()`
This function is written in assembly to reduce the delay between reading TMR1L and TMRLH. Even though this delay is very short, TMR1H must be checked after TMR1H is read to ensure that there is no change in the high order bits of the register during lower byte reading.

`Filter()`
This function filters out noise and unusual signals. If the status of input keeps less than a certain time, the change of signal will be considered noise or as an unusual signal and the output will remain unchanged.

6. Compensation for Speed Calculation

From the field tests, we found a problem that caused the speed of one side of a vehicle slightly faster than those of the other side. Although it was not a big difference, it existed persistently.

After doing a large number of experiments and tests, we eventually found the sensitive area of 4 detection elements of one photodiode array was not uniformly distributed on the projected laser line as shown in Figure 6-1. Because our speed calculation is based on the assumption that the physical geometric location of the detection elements are uniformly distributed, such uneven distribution induced a tiny error in the calculation results. Since two sensitive points P1 and P2 on projected laser line 1 were lower than those on projected laser line 2. Considering the front vehicle curvature , it is obvious that the time interval for vehicles passing through pair1 and pair2 becomes shorter. We assume that the distance of two projected laser lines is D_s and the actual speed of vehicles is V_r. The calculated speed V_c is

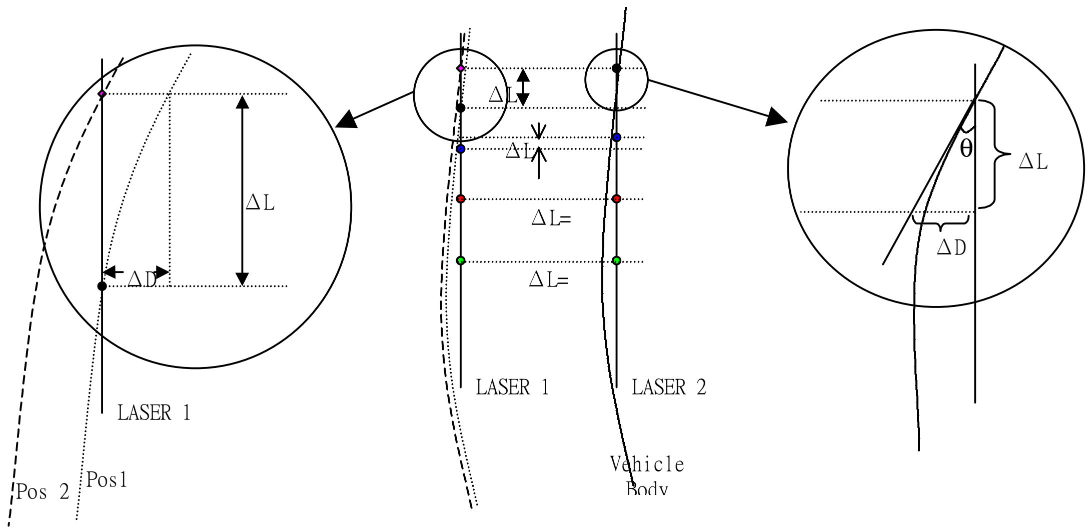


Figure 6-1 Compensation for Speed Calculation

$$V_c = \frac{Ds}{\Delta t}$$

$$\Delta t = \frac{Ds - \Delta D}{Vr} = \frac{Ds - \Delta L \sin\theta}{Vr}$$

$$Vc = \frac{Ds}{\Delta t} = \frac{Ds}{Ds - \Delta L \sin\theta} Vr$$

$$\therefore Vr = Vc \frac{Ds - \Delta L \sin\theta}{Ds} = Vc(1 - \frac{\Delta L \sin\theta}{Ds})$$

If $\Delta L=0$ or $\theta=0$, then Vc=Vr.

But in our system $\delta L1 \neq 0$, $\delta L2 \neq 0$. As a result, the speeds 1 and 2 are faster than v3 and v4.

This problem introduced a new adaptation issue in the system design. In order to determine the correct speed, our system need to be able to map the physical boundary and orientation of the blocking vehicle automatically. We did a detailed evaluation of all plausible geometric orientation which might induce errors and modified source code to compensate for those errors.

7. Mechanical Design

The laser-based detection system we have developed will be used to identify vehicles with high accuracy. The identification data can be used to determine the real travel time of vehicles on the highway. The advantages our system has over the other systems currently used for vehicle identification, such as loop detectors [3], video image processing, and the Schwartz Electro-Optics Autosense III sensor are easy installation, low power consumption, higher reliability, and less computation. Because our system uses active laser light, it produces its own signals to be sensed and does not suffer from limitations of lighting conditions.

In our laser-based system, vehicle lengths (bumper to bumper) are measured by the laser lines to form the vehicle outline profiles. The outline profile is used as the primary identifying feature. The system operates in the following manner**:** 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 utilizing line-generating optics, which project the laser to a flat surface where objects are to be detected. The detector consists of imaging optics and a linear photodiode array. The photodiode array receives the laser light reflected back from the region of detection. The signal from the photodiode is amplified and sent to a computer for processing. Vehicle presence is based on detection of the absence of reflected laser light. Two of these units are integrated and placed a known distance apart, allowing the speed of the object and its residence time under each detector to be measured. This data is used to calculate the object's length and compile a top-down outline profile. The details of the measurement principle are described in [2].

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 time is recorded. When the second beam is blocked, a second time is recorded. These times are used to calculate the front speed of the car. In a similar manner, when each of the beams is no longer blocked, the times are recorded and the rear speed of the vehicle is calculated. The duration that each detector is blocked is also recorded and used to calculate the vehicle length, assuming constant vehicle acceleration. In order to obtain a stable and good image on the sensors, a properly designed mechanical system is required. The mechanics should be rigid enough for field mounting and testing.

The mechanical design for the optics should be flexible and allow the optical components to be adjusted to optimize alignment and focus of the reflected laser beam. In the original system it was impossible to ensure manufacturing accuracy , because there were so many components in the system and the tolerances of some components were unknown. Even though some venders provide tolerances for some components, we were unable to determine the tolerances of ordered optical and laser components before they arrived. Even though we could get signals from this kind of optical-mechanical system, the chance of having an optimized system was very low. Thus the system did not provide a good image or necessary critical height. .
There are some diffractive light and/or high-order peaks outside the width of the Gaussian profile. These diffractive side beams are hard to see, but we did observe a high order laser beam beside the main laser beam after using an IR detection card. The side laser beams are much lower than the laser peak. If the vehicle has high reflectance, significant reflection can still occur from this component of the beam. This reflection can be comparable to the reflection from the road.

Accuracy is not achieved in manufacturing. All the components are adjustable in two or three-dimensions. The components are held by springs. Using these kinds of conventional opto-mechanics, stability and accuracy are easily achieved without precise manufacturing. The cylindrical lenses, imaging lenses, and APD sensors are mounted on optical rails which are mounted to the base plate. Each optical component and sensor is clamped in a mount, which is adjustable in two dimensions. The imaging lenses can be swung around an axis with the optical cell. This allows the system to be adjusted until the axis of the telescope and imaging lens are aligned.

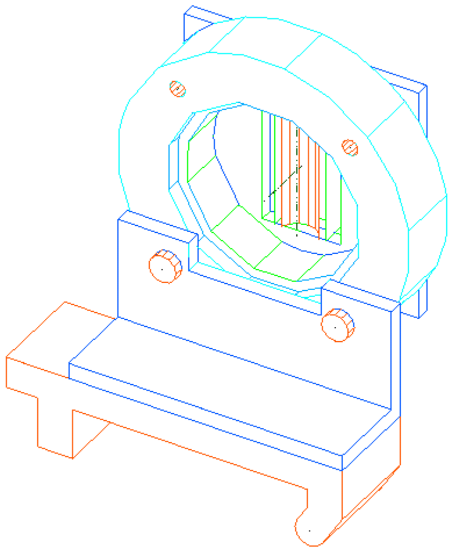


Figure 7-1 Sensor Mounting.

The sensor is clamped by a ring mounted to a rotating lens holder, as shown in Figure 7-1. As a result the sensor can be adjusted through translation and rotation. This mechanism ensures the linear sensor array is perpendicular to the laser line image and is in the exact same position as the laser line. The mechanical mounting for the imaging lens is shown in Figure 7-2. The lens can swing about the X and Y-axis. The whole unit is mounted to a slide which is clamped to the rail. The lens can also be adjusted in the X direction to match the axis of the telescope.
Another area of concern is the structural rigidity of our system under prolonged mounting conditions. In favor of reducing the weight of our system, most components are aluminum. The only part that does not utilize aluminum is the structural backbone of our system. Carbon steel is used here to provide support for the plate which the system is mounted on, and to add support for the mountings. The relative positions of the mountings are found on top of this backbone, where bolts extend through both the plate and structural members. Support for the plate is crucial since any bending and twisting of the plate caused by environmental influence (i.e. wind) can greatly affect data accuracy. All system components are mounted to a metal plate.
The laser projected down to the road (9 meters) and reflected back (9m)to the system's sensor optics.. Therefore the optical path is very long. Any small change in the relative position of the laser, sensor optics, or different components in the sensor optics will cause a large shift of the image. Therefore the optical system must be adjustable after mounting. The mechanical system must be rigid enough for field use as discussed above, but the system can not be too heavy because it will be mounted over a highway. As shown in Figure 7-3, a frame was designed to support the system. It has been proven stable and the weight has not increased much.

A visible laser and a CCD camera were used to verify the image quality of the optical system. After careful adjustment of the optical system, a clear image can be seen on the screen.

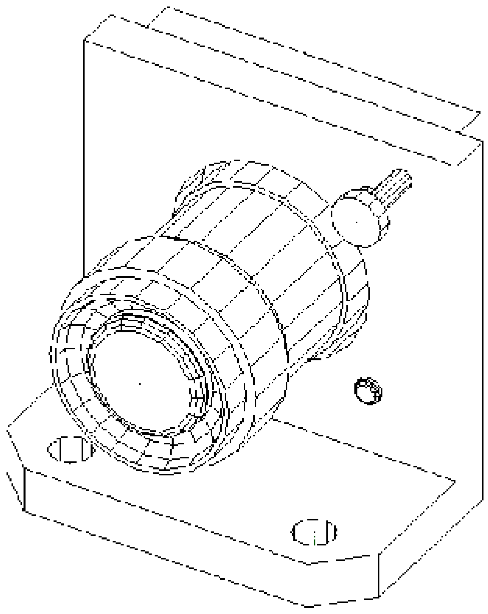


Figure 7-2 Imaging Lens Mounting.

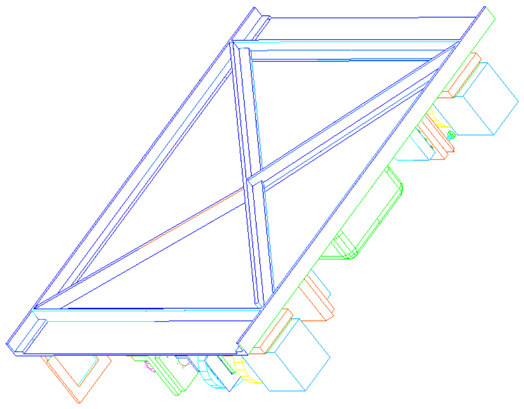


Figure 7-3 The System Support Frame and Plate

The distance between each laser/sensor pair is 60 cm. The sensors are mounted in a fixed vertical position pointing downward, and are focused on the ground forming two detection zones. The lasers are aimed towards the detection zones and mounted at an adjustable angle, allowing the system to be mounted at different heights. In this configuration the minimum detectable object height (critical height) is about 46 cm. This is lower than the bumper height of most common vehicles.

Another design element that might be taken into future consideration is reduction of the back support plate to lower wind stress. This can be done with perforations on the board, or removal of the board completely. Solely using the steel frame had been considered, but mounting problems could not be solved. One solution might be to separate the board into three portions and have empty slots between the portions. The three portions would each contain mounts for the main system components. Thus with less surface area, wind stress could be further reduced.

8. Test

The system has been tested outdoors and on the highway (I-80). The procedures to line up the lasers and adjust the electronic system are as follows.

In the lab:

1. Check for a laser beam and for appropriate position on the wall. If the laser line can not be seen, inspect the laser module opening first. If the laser can be seen in the opening, check if the laser line is being blocked by an obstacle. If the laser can still not be seen in the module opening, check the clock circuit and the 12V DC power supply.
2. Increase the aperture of the image lens past half way.
3. Attach the oscilloscope probe to the output of the amplifier (pin 8) and scan the laser by adjusting the screw on the rear of the laser mount until the maximum signal is reached.
4. Reduce the aperture until the width of the pulse is about 100ns and not saturated. .
5. Adjust the transistor bias to the lowest level possible, as long as the 74HC123 can be triggered continuously. The output of the digital signal can be checked in both the output of the 74HC123 or through the user interface of software.
6. Start the data acquisition software. The procedures to run the software can be found in the software section.

Outdoor/highway test:

1. The system should be adjusted in the lab before going to the roof or the highway. The laser still needs to be adjusted slightly in the field because of differing distances.
2. Follow the lab test instructions steps 2-6.

8.1. Outdoor Test

To test the system in different weather conditions, the system was mounted on the roof of Bainer Hall. The analog signal peak, signal level, and temperature were recorded for more than 24 hours.

Data is recorded through a Tektronix oscilloscope. A computer is setup to read data from the oscilloscope via a serial port. Following is the code to read data from the oscilloscope:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <dos.h>
#include <windows.h>

int main() {
       int i = 1;
    char s[50];
       char m[30];
    char *s1 = "ren WFM_DATA.PRN PULSE";
       char *m1 = "ren measure measure";
    char s2[3];

    while(1){
       itoa(i++,s2,10);
       strcpy(s, s1);
       strcpy(m, m1);
       strcat(s, s2);
       strcat(m, s2);
       system("GETWFM<option");
```

```
        system(s);

        system("tl<option2>measure");
        system(m);


        Sleep(300000);
    }

    return 0;
}
```

The program is compiled in Visual C. *GETWFM* and *tl* are example programs provided by Tektronix. They are run by system call, *system*. *GETWFM* gets the waveform and *tl* measures the peak value and temperature from the oscilloscope. The program reads the peak value and waveform every 30 minutes. This interval is specified by the *Sleep* function call. The result is stored in the files for each measurement. Temperature is measured by a k-type thermocouple, through an adapter from FLUKE. The output is in milli-volts corresponding to the temperature.

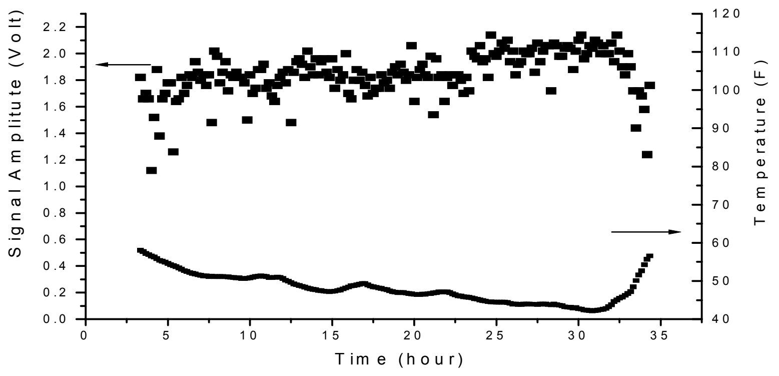


Figure 8-1 Signal Amplitude tested in Clear Weather

Tests were also performed to assess the effects of ambient conditions on system performance. The detection system was mounted outdoors (at a UC Davis test site) for more than 24 hours to test influences of temperature and weather on the system performance. The previous analog system was fairly sensitive to the environment. However, digital circuitry is more reliable and can usually operate over a wide temperature range. Therefore only waveforms and amplitudes of the analog signals in the system were recorded in our experiments. The data were acquired automatically via computer at preset intervals. The signal amplitude and temperature over the long-term test period in clear weather is shown in Figure 8-1. The data were sampled every 5 minutes. As shown in Figure 8-2, the signal amplitude depends mostly on temperature. The signal fluctuation in the figures is due to the unstable nature of the pulsed diode laser, but the trend of a shift in signal amplitude as the temperature changes can still be seen. There was mild fog observed in the morning during testing, but it did not seem to influence the signal amplitude. From Figure 8-2 it can be seen that the signal amplitude increases as the temperature decreases. This is because the sensitivity of the photodiode array increases as temperature decreases. Note that arrows on the graph depict the lower set of points as temperature and the higher set of points as signal amplitude.

The data in Figure 8-3 was recorded once every 10 minutes. The effect from precipitation was minimal, and the signal level was mainly influenced by temperature, similar to the clear-weather results. The noise varied over the range 0.07-0.1 V, which is about one order-of-magnitude smaller than the signal levels. These results verify that the signal fluctuation is not caused by the sensor electronics. Even if the signal fluctuation (noise level) is 0.5 Volts, the signal level is still adequate for triggering the digital output circuitry in a reliable fashion.

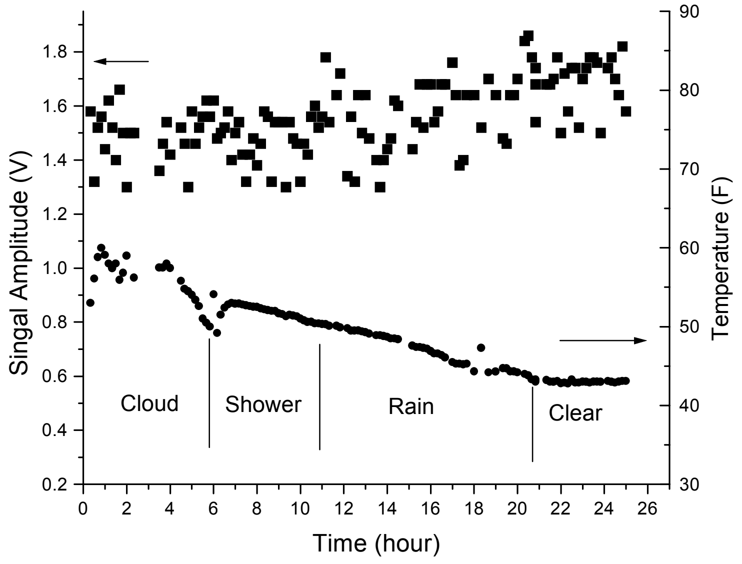


Figure 8-2 Signal amplitude for different weather conditions.

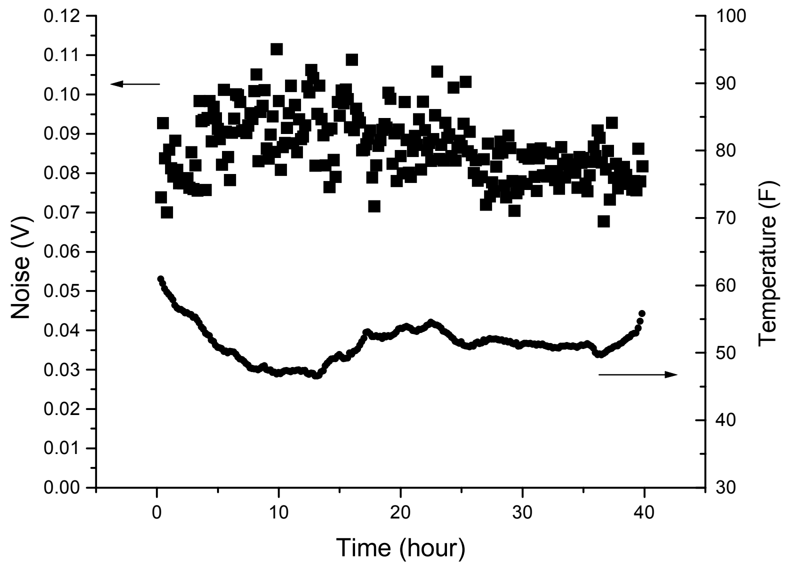


Figure 8-3: Noise levels During an Extended-Time Test.

8.2. Field Test

Field tests have been conducted near the junction of highway I-5 and I-80 in Sacramento. `Figure 8-4` is a picture of the test site and the detection system mounted on the highway overpass. The results Figure 8-5 are from the highway test. A digital video camera was used to record the vehicle passing through the detection system. The picture was downloaded from the camera via an IEEE1394 firewire port. The user interface window is shown on top of each figure. This window contains a menu bar, a status window, and a strip chart. The menu bar is used to control

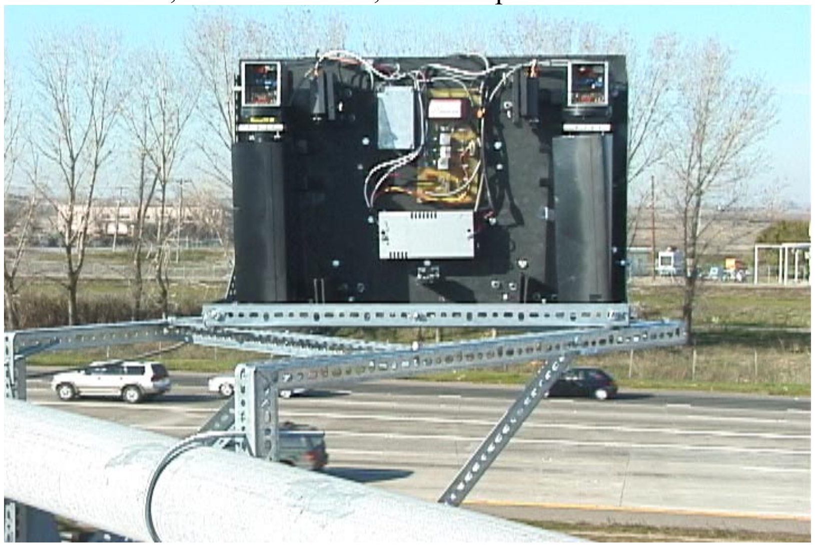
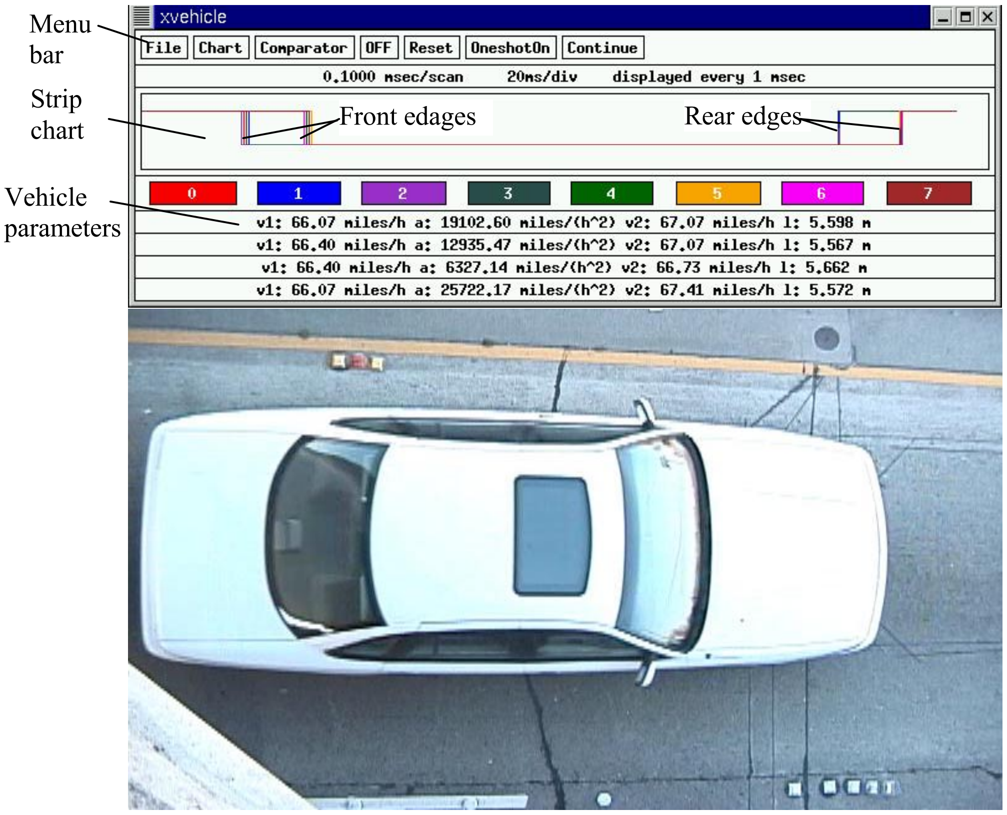


Figure 8-4: Detection System Mounted Above the Highway.

the system, i.e., to start or stop data acquisition and to set parameters of the system. The signals from the electronic sensors are displayed dynamically in the strip chart, which scrolls from left to right. The signal is at a high level when a vehicle is not present, and changes to a low level when a vehicle blocks the laser. This transition occurs over a time that is much shorter than the sampling interval. Since all the signals are displayed at the same position, only one signal line can be seen on the chart when there is a vehicle blocking the laser lines (or no vehicle). Signals from different sensor elements can be distinguished when they change from a low to high level (or inversely) at different times. The front speed (v1), rear speed (v2), acceleration (a), and length (l) of the vehicle are displayed on the lower part of the window. Each line corresponds to one pair of sensor elements. Currently we only display four pairs of sensor elements.
As shown on the left side of the strip chart in `Figure 8-5`, the transition edges for the vehicle's front did not occur at the same time. The differences occur because of the curved front bumper on this vehicle. Different parts of the curved front bumper hit the laser (which is planar) at different times. On the other hand, the rear bumpers are essentially flat in this figure and the transition edges for the rear bumpers (in the right side of strip chart) transit up at nearly the same time. These behaviors were observed with other vehicles that had straight or curved bumpers, or when a vehicle changed lanes. The vehicle did not completely block the signal in the left pair of sensor diodes, so there were many transitions in the signal for this pair (this pair corresponds to the third row of vehicle parameters). The vehicle parameters for other sensor pairs were still accurate.

These results show that the signals for the new field prototype system are clear and the transition is fast enough for measurement. The system is consistent in measuring vehicle lengths and speeds. A vehicle traveling at a representative speed of 65 mile/hour will cross the two laser lines in approximately 17.5 ms. As a result, the error caused by the sampling interval is less than 1.14%. Misalignment of the laser lines on the road might also cause some errors, but such errors can be accounted through system adjustment and calibration.



```
v1: 66.07 miles/h a: 19102.60 miles/(h^2) v2: 67.07 miles/h l: 5.598 m
v1: 66.40 miles/h a: 12935.47 miles/(h^2) v2: 67.07 miles/h l: 5.567 m
v1: 66.40 miles/h a: 6327.14 miles/(h^2) v2: 66.73 miles/h l: 5.662 m
v1: 66.07 miles/h a: 25722.17 miles/(h^2) v2: 67.41 miles/h l: 5.572 m
```

Figure 8-5: Test Results for a Passenger Car.

9. Future work

In the current version of system, we have introduced the microprocessor and potentiometers to adjust the electronic circuitry output. This improvement has greatly decreased the adjustment time. But we still have to do laser adjustment manually, which is difficult because the laser beams are invisible outside. In the future, we plan to use a LCD to display the information of reflected laser beams. The information will help us to adjust laser beams effectively and accurately. We will also try to put the new system design (based on DSP) into practice.

The optics also need to be redesigned. Currently there are 4 separate parts mounted on the mounting plate. Sometimes the optics are not on the conjugate axis and system results are incorrect. Mounting all the optics parts into one drawtube to keep them on the conjugate axis permanently will greatly decrease overall system adjustment time and improve accuracy.

10. Conclusion

We have improved our previous methodology for measurement of true travel time on the highway. A feed-back loop, using microchip control, was introduced into the system. It makes the system easy to use. The mechanical components were improved to increase the precision of the system. Software to display the information or communicate with other research groups was updated. We also analyzed factors affecting the accuracy of the system. We improve the algorithm for calculation of speed and height. The software platform was changed from Windows NT to Linux, which has better real-time ability. We have come to the conclusion that high precision of the laser beam can overcome the lower accuracy of general-purpose parts in the integrated system.

11. References

[1] Palen, J, The Need for Surveillance in Intelligent Transportation Systems --- Part Two, *Intellimotion*, Vol 6, No. 2, 1997, pp. 1--17.

[2] Palen, J, The Need for Surveillance in Intelligent Transportation Systems, *Intellimotion*, Vol 6, No. 1, 1997, pp. 1--10.

[3] Ostland, M., et al., Simple Travel Time Estimation from Single-Trap Loop Detectors, *Intellimotion*, Vol. 6, No. 1, 1997, pp. 4--11.

[4] MacCarley, C. A., Advanced Imaging Techniques for Traffic Surveillance and Hazard Detection, *Intellimotion*, Vol. 6, No. 2, 1997, pp. 6--15.

[5] Malik, J., and Russell, S., Measuring Traffic Parameters Using Video Image Processing, *Intellimotion*, Vol. 6, No. 1, 1997, pp. 6--13.

[6] Coifman, B. "Using Dual Loop Speed Traps to Identify Detector Errors", *Transportation Research Record no. 1683,* Transportation Research Board, 1999, pp 47-58.

[7] Coifman, B., Beymer, D., McLauchlan, P., and Malik, J. "A Real-Time Computer Vision System for Vehicle Tracking and Traffic Surveillance", *Transportation Research: Part C,* vol. 6, no 4, 1998, pp. 271-288.

[8] Zhang, H. M. and Recker W. W. (1999) "On optimal freeway ramp control policies for traffic corridors", *Transportation Research,* B.33(6):471-436

[9] Zhang, H. M. (1999) "A mathematical theory of traffic flow hysteresis," *Transportation Research, B.*33(1):1-23

[10] Palen, J, ITS Roadway Detection Systems Development, personal communication, 1996.

[11] Coifman, B. A.,*Vehicle Reidentification and Travel Time Measurement Using Loop Detector Speed Traps*, Ph.D. Dissertation, University of California, Berkeley, 1998.

[12] Palen, J., Roadway Laser Detector Prototype Design Considerations, personal communication,1996.

[13] Tyburski, R.M., A Review of Road Sensor Technology for Monitoring Vehicle Traffic, *ITE Journal.* Vol. 59, no. 8 (Aug. 1989) Wangler, et al., *Intelligent Vehicle Highway System Sensor and Method*, U.S. Patent No. 5,546,188, 1996.

[14] Olson R. A.; Gustavson R L., Wangler R J., McConnell R E., Active Near-Field Object Sensor and Method Employing Object Classification Techniques, *U.S. Patent* No. 5,321,490, 1994.

[15] Wangler R. J., Gustavson R L., McConnell R E., Fowler K L., Intelligent Vehicle Highway System Sensor and Method*, U.S. Patent* No. 5,757,472, 1998.

[16] Wangler R. J., Gustavson R L., McConnell R E., Fowler K L., Intelligent Vehicle Highway System Multi-Lane Sensor and Method, *U.S. Patent* No. 5,793,491, 1998.

[17] Emerson, L., Mobil Video Surveillance and Ramp-Metering System, *Intellimotion*, Vol 6, No. 2, 1997, pp. 10--12.

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

[19] Cheng H. H., Shaw B. D., Palen J., Larson J. E., Hu X., Katwyk K. V., A Real-Time Laser-Based Detection System for Measurement of Delineations of Moving Vehicles*, CD-ROM Proc. of the ASME 19th Computers in Engineering Conference*, paper # DETC99/CIE-9072, Las Vegas, NV, September 12-15, 1999.

[20] Larson, J. E., Van Katwyk, K., Cheng, H. H., Shaw, B., and Palen, J., *A Real-Time Laser-Based Prototype Detection System for Measurement of Delineations of Moving Vehicles*, California PATH Working Paper, UCB-ITS-PWP-98-20. California PATH Program, Institute of Transportation Studies, University of California, Berkeley, September 1998.

[21] Bin Lin, Harry H. Cheng, Benjamin D. Shaw and Joe Palen, *"A Laser-Based Non-Intrusive Detection System for Real-Time Measurement of Delineations of Moving Vehicles on the Highway",* submitted to Optics and Lasers Engineering, ELSEVIER.
[22] *American National Standard for the Safe Use of Lasers*, Laser Institute of America, Orlando, 1986.
[23] LynxOS Lynx Real-Time Systems, Inc., *LynxOS Application User's Guide*, release 2.2.1, Los Gatos, CA, December 1993.
[24] Alessandro Brubini, Linux Device Drivers, 1st Edition, Oreilly, 1998.
[25] Rubini, Alessandro, *Linux Device Drivers*, First Edition, O'Reilly, 1998.
[26] Pedrotti, F. L. Pedrotti, L. S., *Introduction to Optics*, 2nd Ed., Prentice Hall, New Jersey, 1993.
[27] Ritchie, D. M. and Thompson, K. L., The Unix Time-Sharing System, *Commun. ACM*, vol.~17, No.~7, July 1974, pp.~365--375.
[28] Thompson, K., Unix Implementation, *The Bell System Technical Journal*, vol. 57, No. 6, July-August 1978, pp.~1931--1946.
[29] Kang, S. and S. Ritchie (1998). "Prediction of Short-Term Freeway Traffic Volume Using Recursive Least Squares and Lattice Filtering." Proceedings, 5th International Conference on Application of Advanced Technologies in Transportation, Newport Beach, USA. Edited by C. Hendrickson and S. Ritchie. American Society of Civil Engineers.

12. Appendix:

12.1. Source Code in microchip

laser.h

```
void XdcpGoZero();
void XdcpSet();
void XdcpGoUp();
void XdcpGoupHalf();
void DetectPeriod();
void Init();
void Pwm();
void XdcpIntHandling();
void FlashLed();
void LightLed();
void DarkLed();
void FiltIntHandling();
void ReadTime();
void C2Int();
void Filter();
void WriteByteToEE();
void ReadByteFromEE();
----------------------------------------------------
defreg.h
/**************************************************
 * define variable and special function register
 * only for pic16F876 according to p16F876.inc
 *
 *  by  :  zhaoqing Wang
 *  date:  11/08/2001
 *************************************************/

// bank0 registers
// char INDF      @0x0000; //already defined
// char TMR0      @0x0001; //already defined
// char PCL @0x0002; //already defined
// char STATUS    @0x0003; //already defined
// char FSR @0x0004; //already defined
// char PORTA     @x0005;  //already defined
// char PORTB     @0x0006; //already defined

char PORTC  @0x0007;

// char PCLATH     @0x000A; //already defined
// char INTCON     @0x000B; //already defined
int  TMR1   @0x000E;
char PIR1   @0x000C;
char PIR2   @0x000D;
char TMR1L  @0x000E;
char TMR1H  @0x000F;
char T1CON  @0x0010;
char TMR2   @0x0011;
char T2CON  @0x0012;
char SSPBUF @0x0013;
char SSPCON @0x0014;
char CCPR1L @0x0015;
char CCPR1H @0x0016;
char CCP1CON      @0x0017;
char RCSTA  @0x0018;
char TXREG  @0x0019;
char RCREG  @0x001A;
char CCPR2L @0x001B;
char CCPR2H @0x001C;
char CCP2CON      @0x001D;
char ADRESH @0x001E;
char ADCON0 @0x001F;

// bank1 registers

//char OPTION_REG @0x0081; //already defined
//char TRISA      @0x0085; //already defined
//char TRISB      @0x0086; //already defined
char TRISC      @0x0087; //already defined
char PIE1       @0x008C;
char PIE2       @0x008D;
char PCON       @0x008E;
char SSPCON2    @0x0091;
char PR2        @0x0092;
char SSPADD     @0x0093;
char SSPSTAT    @0x0094;
char TXSTA      @0x0098;
char SPBRG      @0x0099;
char ADRESL     @0x009E;
char ADCON1     @0x009F;

//bank2 register
//char EEDATA     @0x010C; //already defined
//char EEADR      @0x010D; //already defined
char EEDATH     @0x010E;
char EEADRH     @0x010F;

//bank3 register
//char EECON1     @0x018C; //already defined
//char EECON2     @0x018D; //already defined
----------------------------------------------------------------------
laserdef.h
----------------------------------------------------------------------
// for init port a,b,c

#define  TRIS_B 0xF0     //<4:7>pins of port b as interrupt input,<0:3>
pins as output
#define  TRIS_C 0x01     // pin <0> as input, other as output
#define  TRIS_A 0x00     // all pins<0:7> of port a as input


// copy from tt1.asm wroted by zhang

#define OPTION_P  0x4e  //Prescale assign to WDT with 1:64
#define INT_INT_P 0X90  //Enable int/RB0  int
#define RB47_INT_P      0x88  //Enable DB4-7 int
#define PIE1_P          0X00  //Disable all peripheral int
#define PIE2_P          0X00  //Disable other int
```

```
#define PCON_P          0x03  //clr POR/BOR
#define ADCON1_P  0x06  //Set porta as digital I/O

#define PERIOD_FIL1     0X03  //PM tunning filter,contiuously 3 times
period
                        //correct, then PM is OK
#define PERIOD_FIL2     0X10  //Period detect filter
#define T1CON_P         0x31    //Prescale 1:8, enable timer1.P51
/*************************************************
 * the value of T1CON_INIT is calculated as below:
 *      Enable tmr1 with internal clk, prescale=8
 *    max(TMR1L)=256*Tosc*4*8=512us  Tosc=1/16MHz
 *************************************************/
#define T2CON_P         0x06    //Enable TMR2, Prescale=16,
postscale=0. P55

/*=======================================================================
====
 *PWM PARAMETER
 *PWM Period=[(pr2)+1]*4*Tosc*(TMR2 prescale value=16)
 *Tosc=100ns, pr2=26 then PWM period=5000Hz
 *============================================================*/
#define PR2_P           0x26  //0x26

/*=======================================================================
====
PWM duty circle=(CCPR1L:CCP1CON<5:4>).Tosc.(TMR2 prescale valve=16)
  Tosc=100ns, TMR2 Precscal valve=4,
(CCPR1L:CCP1CON<5:4>)=13=0dh=000011.01
  ie: CCPR1L=0X06 & CCP1CON<5:4>=0X01 then PWM duty cir=10us
==============================================================*/

#define CCPR1L_P  0x03
#define CCP1CON_P 0x1C  //Set PWM Mode and clr CCPxX:CCPxY
//#define RBIF          0       //RB0 interrupt

#define DIFF_P          0x03    //the torelance of period
#define PERIOD_P  78     //(0x26+1)*2

#define PERIOD_U  81     //PERIOD_P+DIFF_P
#define PERIOD_D  75     //PERIOD_P-DIFF_P

#define FILT_TIMEOUT    780    //PERIOD_P*10
#define FILT_NUM  21

#define TIMEOUT         156   //~PERIOD_P*2   timeout of interrupt

/*============================================================
   output(interrupt) period
   =[(pr2)+1]*4*Tosc*(TMR2 prescale value=16)/
          ((TMR1 prescale value=8)*4*Tosc(TMR))
   =78, timeout =. 2*period, so TIMEOUT take 150.
=============================================================*/


#define XDCP_FINISHED_NUM 3     //continuous times of adjusting the
xdcp to ensure the output in
```

```
                              //required period.

#define BIT0            0      //set bit
#define BIT1            1
#define BIT2            2
#define BIT3            3
#define BIT4            4
#define BIT5            5
#define BIT6            6
#define BIT7            7

#define BIT_IMG0  0x01
#define BIT_IMG1  0x02
#define BIT_IMG2  0x04
#define BIT_IMG3  0x08
#define BIT_IMG4  0x10
#define BIT_IMG5  0x20
#define BIT_IMG6  0x40
#define BIT_IMG7  0x80

#define LOW_LEVEL 0
#define HIGH_LEVEL      1

#define TRUE            1
#define FALSE           0
/* not supported
enum xdcp_status
{
      XDCP_INIT,  //initial status,the system is just power on
      XDCP_WORK,      //all channels(4 channels) tunning finished
      XDCP_ERR        //one of channels tunning is error
};
*/
#define XDCP_INIT 0
#define XDCP_WORK 1
#define XDCP_ERR  2
#define SPEED_NO  50
#define SPEED_SOME      100
#define SPEED_ALL 200
#define MAX_STEP  100
#define XDCP_ADDR 0x00
#define SPEED_ADDR      0x01

/**********************************************************************
*****/
/*
*/
/* This program is for Microchip PIC16F876 to control X9312
*/
/* Digitally Controlled Potentiometer (XDCP)and data pre-processing
*/
/* of signals from sensor electronics.
*/
/*
*/
/* Functionality:
*/
```

```
/*   1) Set the XDCPs values to achieve proper signal gain of signal
*/
/*      detection circuit.
*/
/*   2) Filter out the spikes and noise from the detcted signals.
*/
/*
*/
/*
*/
/* By : Zhaoqing Wang
*/
/* Date : 08/11/2001
*/
/* Version : 2.0
*/
/*
*/
/* The updated version 2.0. differs from the V1.0 in the algorism of
*/
/* signal processing. In version 1.0, the period is used to distiguish
*/
/* noise from signal. In the version 2.0, the noise is detected by
*/
/* measuring the duration of signal. If the duration of signal is less
*/
/* than a certain amount of time tick, it is considered as noise and
*/
/* will be filtered out.
*/
/*
*/
/* Update date : 08/31/2001
*/
/*
*/
/*
*/
/* A LED which is connected to RC7 is used to indicated the adjustment
*/
/* status of XDCP. If system works normally, it will be on. If there is
*/
/* an error in setting XDCP, it will be blinking. The frequency of
blinking*/
/* depends on the values of XDCP when error occurs.
*/
/*
*/
/* The output of sensors electronics is feeded back to PIC16F876
*/
/* portb <4:7>. The processed signals are output to the host computer
*/
/* from portb <0:3>. The enable signal (cs) is connected to port A
<0:3>   */
/*
*/
```

```
/*
*/
/*********************************************************************
*****/

#include "laser.h"     // Function prototype declearations
#include "laserdef.h"  // Definitions of constants and initial values
of
                       // special function registers
#include "defreg.h"    // Defination of addresses of
                       // all available special function registers

char i;
int j;
char flash_speed;       // Frequency of LED blinking
char temp_port;

char interrupt_time_status[24]; // Array to store time and port status
for 8
                                // interrupts, each interrupt takes 3
bytes.

char int_mem_offset;    //offset of interrupt_time_status array.
char interrupt_handle;  //number of interrupts.

int  read_tmr1;         //return from ReadTime().
char read_tmr1l @0x40;  //low byte of 16-bit time.
char read_tmr1h @0x41;  //high byte of 16-bit time.

char xdcp_cs_buffer;    //select channel of the xdcps.
                        //If the bit is 0, the channel is selected.
char xdcp_ok;           //for xdcp adjusting status

char filt_status;       //Port B status. If the status changed,
                        //the correspoding bit set to 0.

int old_tmr11;          //the time of last interrupt in channel 1
int old_tmr12;
int old_tmr13;
int old_tmr14;

int int_temp;
int temp_int_time;      //Used in function C2Int()

char xdcp_counter;      //counter of going up steps of XDCP.

char old_status;        //last interrupt status in Filter()
char diff_status;       //Exclusive Or between current and old filter
status.

char eedata;
char eeaddress;

/******************************¡*********************************/
/* interrupt handler: interrupt()
*/
/*                                                                    */
/* By : Zhaoqing Wang                                               */
/* date: 08/14/2001                                                 */
/*                                                                  */
/*                                                                  */
/* The handler stores the 16-bit time stamp and portb status        */
/* in an array of 24 bytes, when a interrupt from PortB <4:7> is    */
/* received. The time stamp is stored in the first two byte and     */
/* the status is stored in the third byte. Therefore the array can  */
/* store time stamps and status for 8 interrupts.                   */
/*                                                                  */
/*                                                                  */
/********************************************************************/

void interrupt( void )
{
    ReadTime();

    interrupt_time_status[int_mem_offset] = read_tmr1l;
    int_mem_offset++;
    interrupt_time_status[int_mem_offset] = read_tmr1h;

    int_mem_offset++;
    interrupt_time_status[int_mem_offset] = input_port_b();

    interrupt_handle++;
    int_mem_offset++;
    if (int_mem_offset >=23)
        int_mem_offset = 0;

    //int_mem_offset &= 0x07;  //0x0f;   int_mem_offset mod 16 ;for old
version
    clear_bit( INTCON, RBIF ); // clear portb<4:7> interrupt flag

}

/*************************************************************/
/* XdcpSet()                                                 */
/*                                                           */
/* XDCP will be set to proper value so that laser signal can */
/* be detected by digital output circuit.                    */
/*                                                           */
/* by   : zhaoqing Wang                                      */
/* date : 08/14/2001                                         */
/* update: 08/27/2001                                        */
/*                                                           */
/*************************************************************/

void XdcpSet()
{
    XdcpGoZero();            // XDCP is set to 0.
    delay_s(5);              // Wait for the system to become stable.

    disable_interrupt(GIE);
    i = 0;
    int_mem_offset = 0;
    interrupt_handle = 0;
    xdcp_counter = 0;
    old_status = 0;

    xdcp_cs_buffer = 0x30;  // all channels are selected.
    enable_interrupt(GIE);

    temp_port = input_port_b();
    if ((temp_port & 0xf0) == 0xf0 )
    {
        xdcp_ok = XDCP_ERR;
        flash_speed = SPEED_NO;
        return;
    }

    while( TRUE )
    {
        if ( xdcp_cs_buffer == 0x3f ) // all potentiometers are ready.
        {
            xdcp_ok = XDCP_WORK;      // set the LED as work status.

            /* 0x30 -> 0x35: the 4th and 6th channels are too sensitive
to triggle the signal before
                5th and 7th.So here reduce the 4th and 6th
potentiometers one step(100 ohm) */
            /* edit on 12/13/2001 by zhaoqing (binary:0011)*/
            /* 4567->(0312)*/
            /*output_port_a( 0x33 );    // add last 100 ohm. */
            /*output_port_a( 0x30 );      // resotre to the original
value
            XdcpGoUp();
            */

            /*delay_us(200); delay_us(200);*/
            output_port_a( 0x3f );    // store wiper position.

            break;
        }

        if( interrupt_handle>0 )      // At least one channel is ready.
            XdcpIntHandling();

        xdcp_counter++;
        if (xdcp_counter >= MAX_STEP) // xdcp_counter>100 steps
indicates an error.
        {
           xdcp_ok = XDCP_ERR;        // set the LED as error status.
/*
           output_port_a( xdcp_cs_buffer );    // add last 100 ohm for
good channels.
           XdcpGoUp();
           delay_us(200);delay_us(200);
*/
           output_port_a( 0x3f );              // store wiper position

           if (xdcp_cs_buffer == 0x30)
               flash_speed = SPEED_ALL;
           else
               flash_speed = SPEED_SOME;
           break;
        }

        output_port_a( xdcp_cs_buffer);

        XdcpGoUp();
        delay_ms(1);
//delay_us(200);delay_us(200);
        output_port_a( 0x3f);

    }
}

/*******************************************************************/
/* XdcpIntHandling()                                               */
/*                                                                 */
/* When the adjust button is pushed, it generates an interrupt and.*/
/* the interrupt counter is incremented. This function is called   */
/* by XdcpSet() when the interrput counter is not 0.               */
/*                                                                 */
/* This function reads data from interrupt status array and        */
/* dtermine which channel(s) cause the interrupt and set the       */
/* channel selection varible xdcp_cs_buffer.                       */
/*                                                                 */
/* by   :  zhaoqing wang                                           */
/* date :  08/14/2001                                              */
/* update :08/27/2001                                              */
/*                                                                 */
/*                                                                 */
/*******************************************************************/

void XdcpIntHandling()
{
     char temp_int_status;

     i++;
     i++;       // first 2 char is for time stamp, the 3rd is for portb
status
     temp_int_status = interrupt_time_status[i];
     diff_status = old_status ^ temp_int_status;
     old_status = temp_int_status;
     i++;
     if( i>=23 )
         i = 0;

     // for channel 1
     if( diff_status & BIT_IMG4 )        // channel 1 generates the
interrupt.
     {
         if (old_status & BIT_IMG4)          // stop adjusting.
             set_bit(xdcp_cs_buffer,BIT0);   // clear the bit of
channel selection varible.
     //    else
     //        clear_bit(xdcp_cs_buffer,BIT0); // low level means keep
going up a step
     }
     // for channel 2
     if( diff_status & BIT_IMG5 )
     {
         if (old_status & BIT_IMG5 )
             set_bit(xdcp_cs_buffer,BIT1);
     //     else
     //        clear_bit(xdcp_cs_buffer,BIT1);
     }

     // for channel 3
     if( diff_status & BIT_IMG6 )
     {
         if( old_status & BIT_IMG6)
             set_bit(xdcp_cs_buffer,BIT2);
     //      else
     //        clear_bit(xdcp_cs_buffer,BIT2);
     }

     // for channel 4
     if( diff_status & BIT_IMG7 )
     {
         if( old_status & BIT_IMG7 )
            set_bit(xdcp_cs_buffer,BIT3);
     //      else
     //         clear_bit(xdcp_cs_buffer,BIT3);
     }
}

/*******************************************************/
/* FlashLed()                                          */
/*                                                     */
/* output a pulse to the Port C pin 7 and              */
/* the LED will blink                                  */
/*                                                     */
/* by   : zhaoqing Wang                                */
/* date  : 08/14/2001                                  */
/*                                                     */
/*******************************************************/

void FlashLed()
{
    output_low_port_c(BIT7);
    delay_s(1);
    output_high_port_c(BIT7);
    delay_s(1);
}

void LightLed()
{
    output_low_port_c(BIT7);
}

void DarkLed()
{
    output_high_port_c(BIT7);
}
```

```
/*********************************************************************/
/* Pwm()                                                             */
/*                                                                   */
/* Set capture/compare/PWM Regester1 for pulse width modulation(PWM) */
/* Refer to PIC16F87X manual p57 for details                         */
/*                                                                   */
/* by  : zhaoqing wang                                               */
/* date: 08/13/2001                                                  */
/*                                                                   */
/*********************************************************************/

void Pwm()
{
    PR2 = PR2_P;
    CCPR1L = CCPR1L_P;
    set_bit( CCP1CON, BIT4);
    clear_bit( CCP1CON, BIT5);
    T2CON = T2CON_P;
    CCP1CON = CCP1CON_P;
}

/***********************************************************/
/* Init()                                                  */
/*                                                         */
/* This function initializes the Microchip 16f876 system   */
/*                                                         */
/* By :   zhaoqing wang                                    */
/* date :   08/11/2001                                     */
/*                                                         */
/***********************************************************/

void Init()
{

    ADCON1 = ADCON1_P;
    OPTION_REG = OPTION_P;
    PIE1 = PIE1_P;
    PIE2 = PIE2_P;
    PCON = PCON_P;
    T1CON = T1CON_P;

    set_tris_a( TRIS_A );       // set Port A as output.
    set_tris_b( TRIS_B );       // set port B <7:4> as input of
interrrupt.
                                // set port B <0:3> as output.
    output_port_b( 0x00 );
    set_tris_c( TRIS_C );       // set RC0 as input and others as
output.

    enable_interrupt( RBIE );   // enable RB port change interrupt bit
    disable_interrupt( GIE );   // enable global interrupt bit
    interrupt_handle = 0;
    int_mem_offset = 0;
    old_status = 0;
    filt_status = 0xff;
    i = 0;
    j = 0;

    temp_port = input_port_b(); // output of portb<0:3> are same as
portb<4:7>
    temp_port >>= 4;
    output_port_b( temp_port );
    Pwm();
    eeaddress = XDCP_ADDR;    //read xdcp_ok from eeprom
    ReadByteFromEE();
    xdcp_ok = eedata;
    eeaddress = SPEED_ADDR;     // read falsh_speed from eeprom
    ReadByteFromEE();
    flash_speed = eedata;
    enable_interrupt( GIE);
}

/************************************************************/
/* XdcpGoZero()                                             */
/* This function set the values of all potentiometers to 0. */
/*                                                          */
/************************************************************/
void XdcpGoZero()
{
    output_port_a( 0x20 );  // output port A as 00100000
                            //enable the X9312(XDCP) Up/Down=Down

    for  (i = 0; i < MAX_STEP; i++ )
    {
       output_low_port_a( BIT5 );
       nop();nop();nop();nop();nop();nop();nop();nop();nop();
       nop();nop();nop();nop();nop();nop();nop();nop();nop();
       output_high_port_a( BIT5 );
       nop();nop();nop();nop();nop();nop();nop();nop();
       nop();nop();nop();nop();nop();nop();nop();nop();
    }
    output_port_a( 0x3f );   //disable xdcp tune
}

/*************************************************************/
/* XdcpGoUp()                                                */
/*                                                           */
/* This function increments XDCP by one step.                */
/*                                                           */
/* XDPC must be enabled and set to up before calling this    */
/* function, and disabled after calling this function.       */
/*                                                           */
/* by  : zhaoqing wang                                       */
/* date : 08/14/2001                                         */
/*                                                           */
/*************************************************************/
void XdcpGoUp()
{
    output_low_port_a( BIT5 );
    nop();nop();nop();nop();nop(); //this keeps high level for a
certain time.
    nop();nop();nop();nop();nop();
    nop();nop();nop();nop();nop();
    nop();nop();nop();nop();nop();
    output_high_port_a( BIT5 );
    nop();nop();nop();
    nop();nop();nop();nop();nop();
    nop();nop();nop();nop();nop();
}

/**************************************************************/
/* XdcpGoUpHalf()                                             */
/*                                                            */
/* Adjust the petendiometer to the half of its maxiimum value */
/*                                                            */
/* by  : zhaoqing Wang                                        */
/* date : 08/14/2001                                          */
/*                                                            */
/**************************************************************/
/*
void XdcpGoUpHalf()
{
    output_port_a( 0x30 );  // output port_a as 00110000
                            //enable the X9312(XDCP) Up/Down=Up
                            //XDCP:X9312 Digitally Controlled
Potentiometer
    for( i=0; i< 50; i++ )
    {
       output_low_port_a(BIT5);
       nop();nop();nop();nop();nop();
       output_high_port_a(BIT5);
       nop();nop();nop();
    }
    output_port_a( 0x3f );   //disable xdcp tune
}
*/

/*******************************************************************/
/* DetectPeriod()                                                  */
/*                                                                 */
/* This function detects pulse period. The period is obtained      */
/* by averaging 8 successive periods. One period is obtained from  */
/* 4 interrupts.                                                   */
/*******************************************************************/

/*
void DetectPeriod()
{
    int time_sum;
    period_int_num = 0;         // clear the times
    DETECT_INT = 1;           // set the detecting flag
    TMR1L = 0;                  // clear timer1
    enable_interrupt(INTE);
    enable_interrupt(GIE);
    while( TRUE )
    {
        if( period_int_num == 32)
        {
            disable_interrupt(INTE); //diable the RB0 interrupt
            break;
        }
    }
    DETECT_INT = 0;
    time_sum = period_time[1]-period_time[0];
    for ( i=1; i<=7; i++ )
    {
        time_sum = time_sum + (period_time[i] - period_time[i-1]);
    }
}
*/

/**********************************************************/
/* ReadTime()                                             */
/*                                                        */
/* This function is written by assembly to redueces       */
/* the delay betwwen reading tmr1l and tmr1h.             */
/* Even this delay is very short, TMR1H must be checked   */
/* after TMR1H is read to ensure that there is not change */
/* of time when the lower byte was read.                  */
/*                                                        */
/*                                                        */
/* by : Zhaoqing Wang                                     */
/* date : 08/19/2001                                      */
/**********************************************************/
void ReadTime()
{
    asm {
        movf   TMR1H,W
        movwf  _read_tmr1+1
        movwf  _read_tmr1h
        movf   TMR1L,W
        movwf  _read_tmr1
        movwf  _read_tmr1l
        movf   TMR1H,W
        subwf  _read_tmr1+1,w
        btfsc  STATUS,Z
        return
        movf   TMR1L,W
        movwf  _read_tmr1
        movwf  _read_tmr1l
        movf   TMR1H,W
        movwf  _read_tmr1+1
        movwf  _read_tmr1h
        return
    }
}

/****************************************************************/
/* Filter()                                                     */
/*                                                              */
/* This function filters out noise and unusal signals.          */
/*                                                              */
/* If the status of input keeps less than certain time,         */
/* The change of signal will be considered as noise or unusal   */
/* signal, and the output will remain unchanged.                */
/*                                                        */
/* by  : zhaoqing wang                                          */
/* date : 08/19/2001                                                */
/* update: 08/27/2001                                               */
/*                                                                  */
/******************************************************************/
void Filter()
{
    if ( interrupt_handle > 0 )
    {
        interrupt_handle--;
      FiltIntHandling();
    }

    ReadTime();
    // for channel 1
    if ((filt_status & BIT_IMG0) == 0)
    {
        int_temp = read_tmr1-old_tmr11;
        if( int_temp > FILT_TIMEOUT )
        {
            if(old_status & BIT_IMG4)
                output_high_port_b( BIT0 );
            else
                output_low_port_b( BIT0 );
            set_bit(filt_status,BIT0);
        }
    }


    // for channel 2
    if ((filt_status & BIT_IMG1) == 0)
    {
        int_temp = read_tmr1-old_tmr12;
        if( int_temp > FILT_TIMEOUT )
        {
            if( (old_status & BIT_IMG5) == 0 )
                output_low_port_b( BIT1 );
            else
                output_high_port_b( BIT1 );
            set_bit(filt_status,BIT1);
        }
    }


    // for channel 3
    if ((filt_status & BIT_IMG2) == 0)
    {
        int_temp = read_tmr1-old_tmr13;
        if( int_temp > FILT_TIMEOUT )
        {
            if( (old_status & BIT_IMG6) == 0 )
                output_low_port_b( BIT2 );
            else
                output_high_port_b( BIT2 );
            set_bit(filt_status,BIT2);
        }
    }
    // for channel 4
    if ((filt_status & BIT_IMG3) == 0)
    {
        int_temp = read_tmr1-old_tmr14;
        if( int_temp > FILT_TIMEOUT )
        {
            if( (old_status & BIT_IMG7) == 0 )
                output_low_port_b( BIT3 );
            else
                output_high_port_b( BIT3 );
            set_bit(filt_status,BIT3);
        }
    }


}

/***********************************************/
/*  function name: C2Int()                     */
/*      it converts 2 char to 1 int            */
/*  input:    char: read_tmr1l                 */
/*                  read_tmr1h                 */
/*  output:    int: temp_int_time              */
/*                                             */
/*  by :  zhaoqing wang                        */
/* date:  08/28/2001                           */
/*                                             */
/***********************************************/

void C2Int()
{
 asm{
      movf  _read_tmr1l,W;
      movwf _temp_int_time;
      movf  _read_tmr1h,W;
      movwf _temp_int_time+1;
    }
}

/****************************************************/
/*  function name: FiltIntHanding()                 */
/*                                                  */
/*  by  : Zhaoqing Wang                             */
/* date : 08/28/2001                                */
/* update from the laser.prj   (inter.c)            */
/*                                                  */
/****************************************************/
void FiltIntHandling()
{

     char temp_int_status;

     read_tmr1l = interrupt_time_status[i];
     i++;
     read_tmr1h = interrupt_time_status[i];
     i++;
     temp_int_status = interrupt_time_status[i];
     i++;
     C2Int();
     if( i >= 23)
         i = 0;
     diff_status = old_status ^ temp_int_status;
     old_status = temp_int_status;

     // channel 1
     if( diff_status & BIT_IMG4 )
     {
         old_tmr11 = temp_int_time;
         clear_bit(filt_status,BIT0);
     }

     // channel 2
     if( diff_status & BIT_IMG5 )
     {
         old_tmr12 = temp_int_time;
         clear_bit(filt_status,BIT1);
     }

     // channel 3
     if( diff_status & BIT_IMG6 )
     {
         old_tmr13 = temp_int_time;
         clear_bit(filt_status,BIT2);
     }

     // channel 4
     if( diff_status & BIT_IMG7 )
     {
         old_tmr14 = temp_int_time;
         clear_bit(filt_status,BIT3);
     }

}

/*********************************************************/
/* Reads one byte from the EEprom at the specified address */
/* and returns it                                          */
/*********************************************************/
void ReadByteFromEE()
{
    asm{
        bcf    STATUS,RP1      ;bank0
        movf   _eeaddress,W      ; write address
        bsf    STATUS,RP1
        bcf    STATUS,RP0      ;BANK2
        movwf  EEADR           ;Read from this address

        bsf    STATUS, RP0        ;bank3
        bcf    EECON1, EEPGD      ;Point to EE memory
        bsf    EECON1, RD         ;Initiate a read cycle

        bcf    STATUS,RP0
        movf   EEDATA,W           ;Fetch byte from dataregister
        bcf    STATUS,RP1
        bcf    STATUS,RP0
        movwf  _eedata
    }
}

/*******************************************************/
/* Writes one byte to the EEprom at the specified address */
/*******************************************************/
void WriteByteToEE()
{
    asm{
        bcf    STATUS,RP1
        bcf    STATUS,RP0     ; bank0
        movf   _eedata,W      ; Data to write
        bsf    STATUS,RP1     ; bank 2
        movwf  EEDATA
        bcf    STATUS,RP1     ;bank0
        movf   _eeaddress,W   ;Address to write to
        bsf    STATUS,RP1     ; bank 2
        movwf  EEADR

        bsf    STATUS,RP0     ; bank3
        ;label1
        ;btfsc  EECON1,WR      ;wait for write to complete
        ;goto   label1

      bsf    EECON1,WREN    ;Enable writes to the EEProm
        bsf    STATUS,RP0
        bcf    STATUS,RP1     ;bank1
      bcf    INTCON,GIE     ;Disable interrupts during write

        bsf    STATUS,RP1     ;bank3
        movlw  0x55
        movwf  EECON2         ;write 55h to EECON2
        movlw  0xAA
        movwf  EECON2         ;Initiate a write cycle

      bsf    EECON1,WR      ;start write operation
      bcf    EECON1,WREN     ;       // Disable writes to EEProm
      bsf    INTCON,GIE     ;Disable interrupts during write

    }
}


/**************************************************/
/* the main program only for test                 */
/*                                                */
/*    By : zhaoqing Wang                          */
/*   date: 08/11/2001                             */
/*   waiting interrup and output portc<2> circlely   */
/*                       1<--->0                   */
/**************************************************/

main()
{
    delay_s(5);         // if system comes to steady, it start to work.
    Init();
    j = 0;

    enable_interrupt(RBIE);
    enable_interrupt(GIE);
    old_tmr11 = 0;
    old_tmr12 = 0;
    old_tmr13 = 0;
    old_tmr14 = 0;
    TMR1 = 0;
    while ( TRUE )
    {
                        // for test skip XdcpSet(), only run the
Filter()
        if (input_pin_port_c( BIT0 ) == LOW_LEVEL )
        {

            delay_ms(100); //eliminated trembling
            if (input_pin_port_c( BIT0 ) == LOW_LEVEL )
            {
                XdcpSet();    // adjust XDCP
                disable_interrupt( GIE );

                temp_port = input_port_b(); // output of portb<0:3> are
same as portb<4:7>
                temp_port >>= 4;
                output_port_b( temp_port );
                eeaddress = XDCP_ADDR;
                eedata = xdcp_ok;
                WriteByteToEE();   // write the status into flash mem
                eeaddress = SPEED_ADDR;
                eedata = flash_speed;
                WriteByteToEE();
                old_tmr11 = 0;
                old_tmr12 = 0;
                old_tmr13 = 0;
                old_tmr14 = 0;
                filt_status = 0x00;
                TMR1 = 0;
                interrupt_handle = 0; //init the interrupt
                int_mem_offset = 0;
                i = 0;
                old_status = input_port_b();
                enable_interrupt( GIE );
            }
         }
        else
            Filter();
      switch( xdcp_ok )
        {
          case XDCP_ERR:
              int_temp = 100 * flash_speed;
              if (j <=  int_temp)
                  LightLed();
              else
              {
                  DarkLed();
                  int_temp = 100*(flash_speed+flash_speed);
                  if( j >= int_temp )
                     j = 0;
              }
              j++;
          case XDCP_WORK:
              output_low_port_c(BIT7); //LightLed();
          default:
              output_high_port_c(BIT7); //DarkLed();
        }
    }
}

/*****************
    EOF
*****************/
```

12.2. Source Code of device driver in Linux

DIO96.h

```
/*++
 * Module Name:
 *
 *    DIO96.h
 *
 * Abstract:
 *
 *    Define National Instruments PC-DIO-96 board address and
registers.
 *
*
 * Environment:
 *
 *    RTX application.
 *
 * Revision History:
 *
--*/

//#define IRQ            0x05  // Interrupt vertor for NI
//#define BASE_ADDR      0x180 // Base address of the NI
//#define PTL(x)  (*(PLONG)(x))
/* PPI A */

#define VENDOR_NI         0x1093
#define DIO96_ID          0x0160
#define BUFSIZE           400
//#define HALF_BUFFER_SIZE BUFFER_SIZE/2

#define BASE_ADDR     baseptr1
#define PORTA_ADDR      BASE_ADDR + 0x00 // Port A, aka. Port 0
```

```
#define PORTB_ADDR      BASE_ADDR + 0x01 // Port B, aka. Port 1
#define PORTC_ADDR      BASE_ADDR + 0x02 // Port C, aka. Port 2
#define CNFG_ADDR BASE_ADDR + 0x03 // Config Register

/* PPI B */
#define B_PORTA_ADDR    BASE_ADDR + 0x04 // PPI B Port A, aka. Port 3
#define B_PORTB_ADDR    BASE_ADDR + 0x05 // PPI B Port B, aka. Port 4
#define B_PORTC_ADDR    BASE_ADDR + 0x06 // PPI B Port C, aka. Port 5
#define B_CNFG_ADDR     BASE_ADDR + 0x07 // PPI B Config Port

/* PPI C */
#define C_PORTA_ADDR    BASE_ADDR + 0x08 // PPI C Port A, aka. Port 6
#define C_PORTB_ADDR    BASE_ADDR + 0x09 // PPI C Port B, aka. Port 7
#define C_PORTC_ADDR    BASE_ADDR + 0x0A // PPI C Port C, aka. Port 8
#define C_CNFG_ADDR     BASE_ADDR + 0x0B // PPI C Config Port

/* PPI D */
#define D_PORTB_ADDR    BASE_ADDR + 0x0D // PPI D Port B, aka. Port 10
#define D_PORTC_ADDR    BASE_ADDR + 0x0E // PPI D Port C, aka. Port 11
#define D_CNFG_ADDR     BASE_ADDR + 0x0F // PPI D Config Port

/* Counter/Timer */
#define CLOCKA          BASE_ADDR + 0x10 // Clock or Counter 0
#define CLOCKB          BASE_ADDR + 0x11 // Clock or Counter 1
#define CLOCKC          BASE_ADDR + 0x12 // Clock or Counter 2
#define CLOCK_CTRL      BASE_ADDR + 0x13 // Clock or Counter Control

/* Interrupt control */
#define INTR_CTRL1      BASE_ADDR + 0x14 // First interrupt control reg
#define INTR_CTRL2      BASE_ADDR + 0x15 // Second interrupt control
reg


//#define STOP_TIME     5     // Minutes for shutdown handlers to wait
after
                        // a stop - 0 for indefinite.


/*
typedef struct {
//    LONG  Pid;        // Process ID of sender
      LONG    BufFull;
      LONG  Ack;        // Server acknowledge flag
    UCHAR   Buffer[BUFFER_SIZE];
} MSGSTR, *PMSGSTR;

*/
```

---

control.h

```
#define DATA1 0
#define DATA2 1
#define COMMAND_FIFO 2
#define TASK_CONTROL_FIFO 3
#define SEMEPHORE_FIFO 4
```

```
#define START_TASK      1
#define STOP_TASK 2
#define TICK_RESOLUTION 1e6
#define DIO96_CONF      0x82

#define FIFO_0_READY    1
#define FIFO_1_READY    0

struct msg_struct {
      short int command;
      unsigned char conf;
//    int task;
      int period;
      int buffer_size;
};
```

dio96_module.c

```
/*******************************************************************
 * dio96_module.c
 * Drive driver for DIO-96 digital I/O board in RTLinux
 * Last modified: 05/28/2001
 *******************************************************************/
#include <linux/stddef.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/pci.h>
#include <linux/config.h>
#include <linux/vmalloc.h>
#include <asm-i386/io.h>
#include <linux/errno.h>
#include <rtl.h>
#include <rtl_sched.h>   /*time.h is included in this file*/
#include <rtl_fifo.h>
#include "DIO96.h"
#include "control.h"

/* disable usage counter */
#undef MOD_INC_USE_COUNT
#undef MOD_DEC_USE_COUNT
#define MOD_INC_USE_COUNT
#define MOD_DEC_USE_COUNT

unsigned char * baseptr0;
unsigned char * baseptr1;
int status[2] = {1,1};
pthread_t *dio96_task;
struct msg_struct msg;
int i;

unsigned long dio96_setup(void);
void *thread_code(void *t);
int msg_handler(unsigned int fifo);
int data_handler(unsigned int fifo);

int init_module(void) {
    int c[5];
    pthread_attr_t attr;
    struct sched_param sched_param;
    int ret;

    /* configure dio96*/
    *baseptr1 = dio96_setup();

    /* create fufo */
    rtf_destroy(0);
    rtf_destroy(1);
    rtf_destroy(2);
//    rtf_destroy(3);
    rtf_destroy(4);
    c[0] = rtf_create(0, BUFSIZE);
    c[1] = rtf_create(1, BUFSIZE);
    c[2] = rtf_create(2, 40);
//    c[3] = rtf_create(3, 20);
    c[4] = rtf_create(4, 20);
    printk("Fifo return 0=%d 1=%d 2=%d 3=%d
4=%d\n",c[0],c[1],c[2],c[3],c[4]);

    /* create a real-time task*/
    dio96_task = vmalloc(sizeof(pthread_t));
    pthread_attr_init (&attr);
    sched_param.sched_priority = 4;
    pthread_attr_setschedparam (&attr, &sched_param);
    ret = pthread_create (dio96_task,  &attr, thread_code, (void *)1);

    /* create fifo handlers */
    rtf_create_handler(0, &data_handler);
    rtf_create_handler(1, &data_handler);
    rtf_create_handler(2, &msg_handler);

    return 0;
}

/*Kernel module thread*/
void *thread_code(void *t) {
    int count = 0;
    short int rtf = 0;
    short int ret;
    int n;
/*
    int i;
    i=0;
*/
    while(1) {
        ret = pthread_wait_np();
        /* check if the FIFO is filled up*/
        if(count == msg.buffer_size) {
            count  = 0;
            status[rtf] = 0;
            n = rtf_put(4, &rtf, 1);
            rtf=(~rtf)&0x01;
        }
//        if(status[rtf] == 0)
//            printk("queue overflow\n");
```

```
        /* write data into the FIFO*/
        rtf_put(rtf, baseptr1, 1);
/*
        if (*baseptr1 != 0xff && *baseptr1 != 0x0)
        {
            printk("%0x ", *baseptr1);
            i++;
            if( i == 16 ){
                printk("\n");
                i=0;
            }
        }
*/
        count++;

    }
    return 0;
}

/* handler for message FIFO*/
int msg_handler(unsigned int fifo)
{
    if ((rtf_get(COMMAND_FIFO, &msg, sizeof(msg))) == sizeof(msg)) {
        rtl_printf("Task: executing the command to task; command:%d
period:%d; \
                    buffer_size: %d\n",msg.command, msg.period,
msg.buffer_size);
            switch(msg.command) {
                case START_TASK:                     /* Start data
reading thread */
                    writeb(msg.conf, CNFG_ADDR);
                    printk("START_TASK \n");
                    pthread_make_periodic_np(*dio96_task, gethrtime(),
msg.period);
                    pthread_wakeup_np(*dio96_task);
                    break;
                case STOP_TASK:                      /* Suspend data
reading thread */
                    printk("STOP_TASK \n");
                    pthread_suspend_np(*dio96_task);
                    break;
                default:
                    rtl_printf("RTL task: bad command\n");
                return 0;
            }
    }
    return 0;
}

/*handler for data FIFOs*/
int data_handler(unsigned int fifo)
{
#ifdef DEBUG
    printk("fifo = %d\n", fifo);
#endif
    status[fifo] = 1;
    return 0;
}


/* detect and config DIO_96 board*/
unsigned long dio96_setup(void) {
    #ifdef CONFIG_PCI
    if(pcibios_present()){
//        unsigned long base_addr1, base_addr0, pci_irq_line, offset;
        u32 base_addr1, base_addr0, pci_irq_line, offset, window_value;
        u8 pci_cmd;
        struct pci_dev *dev = NULL;

      /* find PCI-DIO-96 in PCI slots */
        dev=pci_find_device(VENDOR_NI, DIO96_ID, dev);

        /* Disable Master/IO access, Enable memory access */
        pci_set_master(dev);
        pci_read_config_byte(dev, PCI_COMMAND, &pci_cmd);
        pci_cmd |= PCI_COMMAND_MEMORY;
        pci_cmd &= ~PCI_COMMAND_IO;
        pci_cmd |= PCI_COMMAND_IO;
        pci_cmd |= PCI_COMMAND_MASTER;
        pci_cmd |= PCI_COMMAND_INVALIDATE;
        pci_write_config_byte(dev, PCI_COMMAND, pci_cmd);

        /* get base addresses for Mite and I/O register and IRQ line of
board*/
        base_addr1 = dev->base_address[1];
        base_addr0 = dev->base_address[0];
        pci_irq_line = dev->irq;
#ifdef DEBUG
        printk("IRQ = %d\n", pci_irq_line);
#endif

        /* remap base addresses to virtual memory space*/
        offset = base_addr1 & ~PAGE_MASK;
        base_addr1 &= PCI_BASE_ADDRESS_MEM_MASK;
        base_addr1 &= PAGE_MASK;
        baseptr0 = ioremap(base_addr0, 1024*4);
        baseptr1 = ioremap(base_addr1, 1024*4);
//#ifdef DEBUG
        printk("offset = 0x%x\n", offset);
        printk("base_addr_after_mask = 0x%x\n", base_addr1);
        printk("baseptr0 = 0x%x\n", baseptr0);
        printk("baseptr1 = 0x%x\n", baseptr1);
//#endif

        /* Configure Mite */
        /* the window value should calculated from the base addr before
remapping*/
        window_value = (0xffffff00 & base_addr1) | 0x00000080;
        writel(0x0000aeae, (baseptr0+0x0340));
        writel(window_value, (baseptr0+0x00c0));

        /* Testing: write to the offset 0 of the area */
        writeb(0x80,(baseptr1+0x03));
        writeb(0x00,baseptr1);

#ifdef DEBUG
        printk("*baseptr0 = 0x%x\n", readb(baseptr0));
        printk("*(baseptr0+3) = 0x%x\n", readb(baseptr0+0x03));
        printk("*baseptr1 = 0x%x\n", readb(baseptr1));
        printk("*(baseptr1+3) = 0x%x\n", readb(baseptr1+0x03));
#endif
    }
    else {
        printk("No PCI board detected");
        return -ENODEV;
    }
    #endif
    return *baseptr1;
}

void cleanup_module(void) {
#ifdef DEBUG
    printk("%d\n", rtf_destroy(1));
    printk("%d\n", rtf_destroy(2));
#else
    rtf_destroy(1);
    rtf_destroy(2);
    rtf_destroy(3);
    rtf_destroy(4);
    rtf_destroy(5);
#endif

    /* unmap when we unload the driver */
    iounmap(baseptr0);
    iounmap(baseptr1);

    pthread_delete_np (*dio96_task);
    vfree(dio96_task);

    printk("bye dio96...\n");
    return;
}
```