

California PATH Research Report 2007

# Development and Field Testing of Laser Photodiode Array-Based Vehicle Detection Systems

Harry H. Cheng  
Ben Shaw  
Joe Palen  
Hong Duan  
Stephen S. Nestinger  
Bo Chen  
Ping Feng

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

September 10, 2007

# Contents

<b>Abstract</b> .....	<b>4</b>
<b>1 Introduction</b> .....	<b>5</b>
<b>2 Overview of the Laser-photodiode Based Detection System (LBDS)</b> .....	<b>7</b>
<b>2.1 Architecture of the LBDS</b> .....	<b>9</b>
<b>2.2 Opto-Mechanical System</b> .....	<b>10</b>
2.2.1 New Structure of the Optics System.....	10
2.2.2 New Laser Module.....	11
2.2.3 Use of a Laser Pointer .....	12
<b>2.3 Redesign of the Electronics System</b> .....	<b>13</b>
2.3.1 Microcontroller Based Control System.....	13
2.3.2 Laser Diode Module and the Trigger Circuit.....	14
2.3.3 APD array and its bias voltage supply .....	16
2.3.4 Signal Pre-Processing Circuitry .....	17
2.3.4.1 Three-stage Pre-amplifier .....	18
2.3.4.2 Comparator and Monostable Multivibrator .....	18
2.3.4.3 Hysteresis to Overcome Noise and Signal Amplitude Fluctuation .....	19
2.3.4.4 Digitally Controlled Potentiometer .....	20
2.3.4.5 Timing Window.....	20
2.3.4.6 Pre-board Noise Data Analysis and Data Processing.....	21
2.3.4.7 Arrangement of Printed Circuit Boards in Signal Box.....	22
2.3.5 Power Supply Circuit and DC/DC Converter .....	22
2.3.6 Temperature Compensation for Responsivity of APD .....	23
2.3.7 Data Acquisition and Output.....	24
2.3.7.1 Rabbit Data Acquisition .....	25
2.3.7.2 Output Data Through Ethernet Communication .....	25
2.3.7.3 Output Data Through Wireless Ethernet .....	26
2.3.7.4 Parallel Port Output .....	26
2.3.8 Optimal Self-Calibration.....	27
2.3.9 Remote Control for Trolley through Wireless Ethernet .....	31
2.3.10 Software Design for Rabbit Microprocessor .....	32
<b>2.4 New Design of the Mechanical System</b> .....	<b>33</b>
2.4.1 Breadboard for optical system.....	33
2.4.2 LBDS Housing.....	34
2.4.3 New Mounting Mechanism for Highway Test.....	34
<b>2.5 Software on remote computer</b> .....	<b>34</b>
<b>3 Comparison Study with the Previous System</b> .....	<b>35</b>
<b>3.1 Longer length of the Detection Area</b> .....	<b>35</b>
<b>3.2 24 channels</b> .....	<b>35</b>
<b>3.3 Stronger Anti-noise Ability</b> .....	<b>35</b>
<b>3.4 Flexibility for Different Environments</b> .....	<b>35</b>
<b>3.5 Communication</b> .....	<b>35</b>
<b>3.6 Laser Alignment</b> .....	<b>35</b>
<b>4 Testing</b> .....	<b>36</b>

<b>4.1</b>	<b>Lab Testing</b> .....	<b>36</b>
<b>4.2</b>	<b>Roof Testing</b> .....	<b>37</b>
<b>4.3</b>	<b>Mobile Truss Testing</b> .....	<b>37</b>
<b>4.4</b>	<b>Outdoor Highway Testing</b> .....	<b>39</b>
<b>5</b>	<b>Conclusions</b> .....	<b>41</b>
	<b>References</b> .....	<b>41</b>
	<b>Appendix A: Circuit and Board Diagrams</b> .....	<b>43</b>
	<b>Appendix B: Source Code</b> .....	<b>47</b>
	<b>Source Code in rabbit 3200 for control and communication</b> .....	<b>47</b>
	<b>Source Code of TCP/IP communication in Linux</b> .....	<b>67</b>

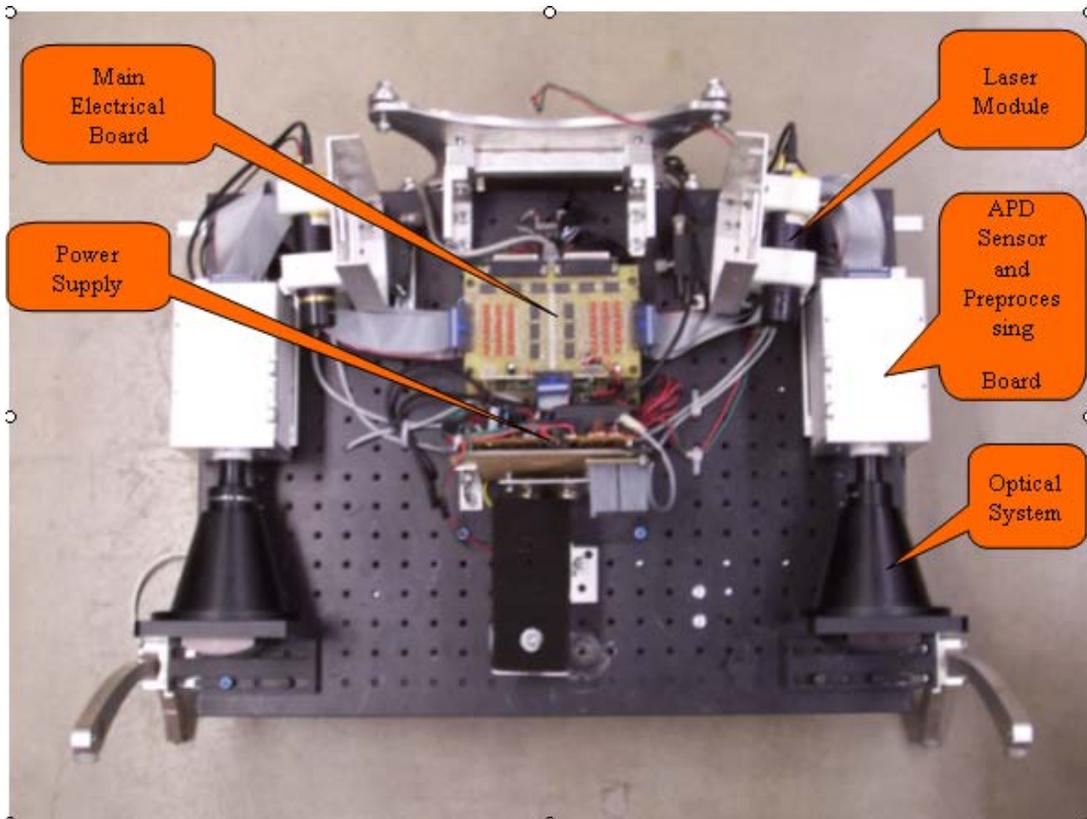
## **Abstract**

Over the past year we have enhanced the performance of the Laser-photodiode Based vehicle Detection System (LBDS) which is sponsored and funded by the California Department of Transportation (Caltrans) through the PATH Program. This project incorporates the development of a vehicle detection system for highway use that can implement correct, reliable, and accurate vehicle detection and has the ability to provide high quality traffic information such as traffic density and vehicle speed, length and profile to the traffic management center for surveillance. After several years of research, the performance of the LBDS has been enhanced and is almost ready for use in practical applications. In the last year, the focus of our research has been on the reliability, accuracy and anti-disturbance capabilities of the LBDS. By using new laser diode modules, the laser line projected on the road surface has been enlarged to 4 m, which almost covers the full width of a lane on the highway, and there is now a uniform intensity distribution along the line, increasing accuracy at the edges of the detection zone. A new optical system has been designed and manufactured that allows the avalanche photodiode (APD) array in the LBDS to “view” the full width of a lane on the highway. Accordingly, the electronic systems have been redesigned. The number of detection channels has been increased from the previous 8 channels to the present 24 channels, increasing the detection resolution. Use of a hysteresis comparator and a timing window has reduced the effect of background noise, enhancing the reliability and stability of the LBDS. Temperature compensation for the APD has been implemented, which stabilizes the amplitude of the analog signal from the APD. Temperature feedback improves system performance over a wide range of temperatures, which afflicted previous designs. The optimal self-calibration capability of the LBDS makes the system more flexible so it can perform in different operating environments. The preprocessing circuit board has been redesigned based on the analysis of the effect of sunlight and oscillating noise on the system. A high-filter circuit was used to increase the S/N ratio which allows for stable long-term use. Outputting the detection data through wireless Ethernet communication makes the LBDS mobile and remotely accessible. With the above improvements on its performance, the LBDS has is now more suitable for the practical application of vehicle detection on the highway. This report provides a detailed description of the design and implementation of the enhanced LBDS. The field-test results of the LBDS prototype at different test sites, including Old Hutchison testing at UC Davis and highway testing in Sacramento, are documented in the report.

# 1 Introduction

Highway traffic information such as the number of vehicles passing through a detection zone, vehicle speeds, and vehicle profiles used for vehicle classification, are fundamental to advanced traffic management systems, e.g., the Intelligent Transportation System (ITS) [1]-[2]. Therefore, the authenticity and accuracy of the information has an adverse affect on the behavior of the ITS. High quality information relies on the performance of field surveillance devices on the highway, i.e., vehicle detectors. Traditionally, highway vehicle detection systems were based around inductive loop detectors [3]-[4]. Although they are still widely used, loop detectors can hardly provide the high quality information required by an ITS [5]. The Laser-photodiode Based vehicle Detection System (LBDS) is a new kind of advanced vehicle detector based on different working principles from the loop detector [6]-[10]; the LBDS has the ability to provide more traffic information with much higher quality compared with loop detectors.

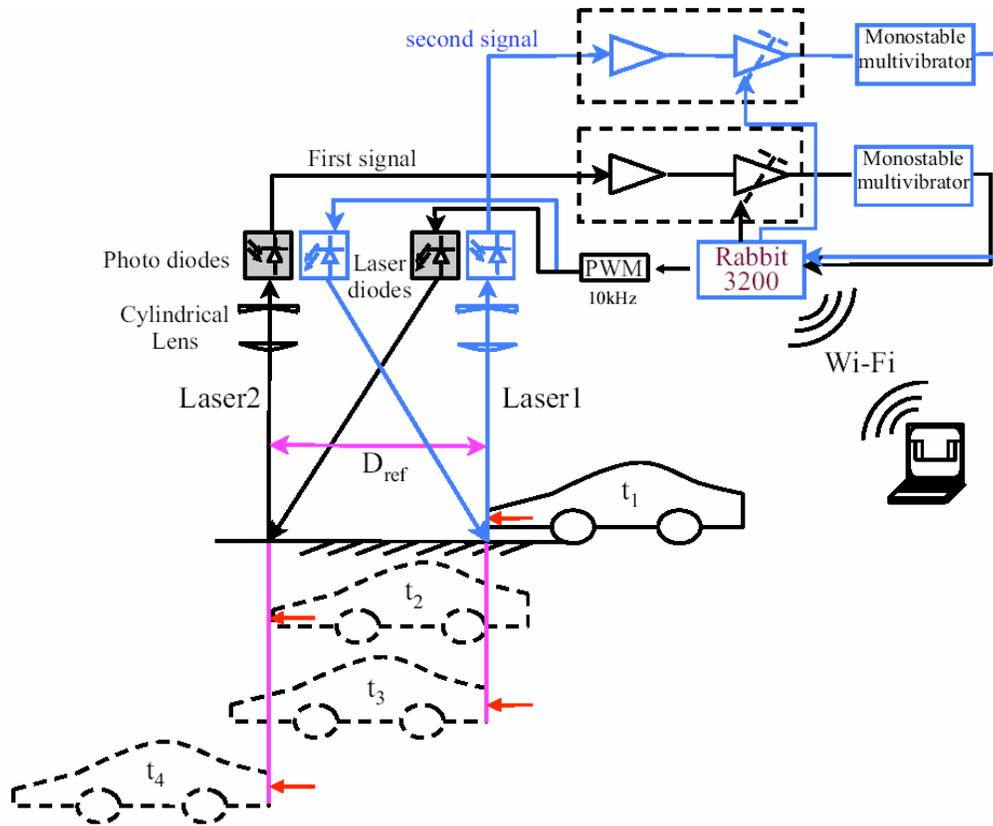
Over the past year we have enhanced the performance of the Laser-photodiode Based vehicle Detection System (LBDS), development of which is sponsored and funded by the California Department of Transportation (Caltrans) through the PATH Program. This project incorporates the development a vehicle detection system for highway use that can implement correct, reliable, and accurate vehicle detection and has the ability to provide high quality traffic information such as traffic density and vehicle speed, length and profile to the traffic management center for surveillance. After several years of research, the performance of the LBDS has been enhanced and is almost ready for use in practical applications. In the last year, the focus of our research has been on the reliability, accuracy and anti-disturbance capabilities of the LBDS. By using new laser diode modules, the laser line projected on the road surface has been lengthened to 4 m, which almost covers the full width of a lane on the highway, and we now have a uniform intensity distribution along the line, increasing accuracy at the edges of the detection zone. A new optical system has been designed and manufactured that allows the avalanche photodiode (APD) array in the LBDS to “view” the full width of a lane on the highway. Accordingly, the electronic systems have been redesigned. The number of detection channels has been increased from the previous 8 channels to the present 24 channels, increasing the detection resolution. Use of a hysteresis comparator and a timing window has reduced the effect of background noise, enhancing the reliability and stability of the LBDS. Temperature compensation for the responsivity of the APD has been implemented, which stabilizes the amplitude of analog signals from the APD. Temperature feedback improves system performance over a wide range of temperatures, which afflicted previous designs. The optimal self-calibration capability of the LBDS makes the system more flexible for use in different operating environments. The preprocessing circuit board has been redesigned based on the analysis of the effect of sunlight and oscillating noise on the system. A high-filter circuit was used to increases the S/N ratio which allows for stable long-term use. Outputting the detection data via wireless Ethernet communication makes the LBDS mobile and remote accessible. With the above improvements on its performance, the LBDS has is now more suitable for the practical application of vehicle detection on the highway. A prototype of a newly designed LBDS is shown in Figure 1-1.



**Figure 1-1. Top view of the 24-channel prototype of the LBDS.**

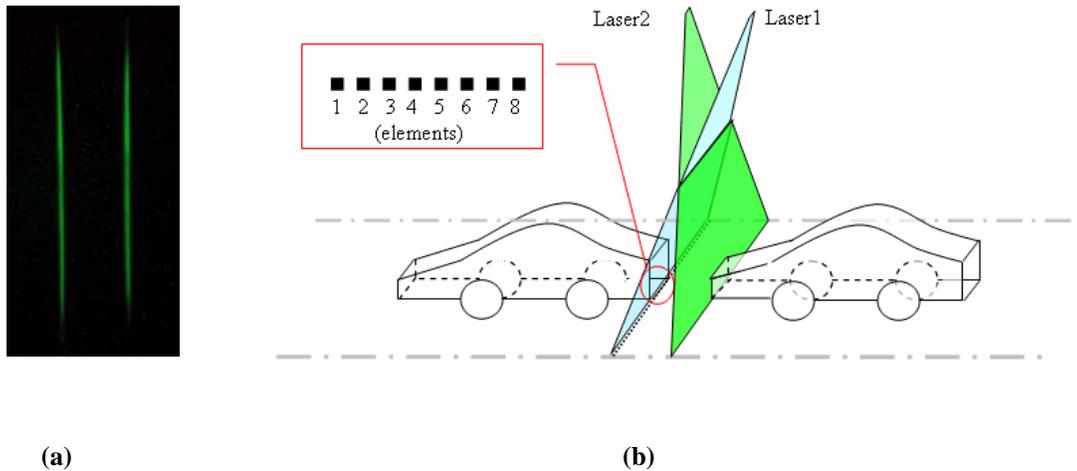
## **2 Overview of the Laser-photodiode Based Detection System (LBDS)**

The LBDS is an overhead vehicle detector that provides the speeds, lengths, widths and profiles of vehicles on the Highway. A diagram of the working principles behind the LBDS is shown in Figure 2-1. The LBDS contains two sets of laser diodes, avalanche photodiode (APD) arrays and optical systems installed in a crosswise fashion. The laser diodes, which are triggered at 10 KHz, project pulsed laser beams onto the road surface. The APD arrays act as laser-sensitive sensors and are used to detect the reflected laser light from the road surface. The optical systems focus the reflected laser light, which creates an image of the projected laser line onto an APD array. When no vehicle is present beneath the LBDS, laser light from the laser diode on left side of the LBDS is reflected from the road surface and is focused onto the APD array on the right side of the LBDS through the right side optical system. Similarly, the laser beam from the right side will be focused onto the left side APD array. Once activated by the reflected laser beam, the APD sensors will output an analog signal to represent the absence of vehicles. Therefore, when the APD arrays receive a laser pulse, the projected laser line is determined to be “unblocked”, i.e., no car is present. When a vehicle is passing under the LBDS, as shown in Figure 2-1, the reflected laser light will be “blocked” by the vehicle, i.e., no light will reach the APD arrays. Therefore, the APD arrays will not output a signal, which represents the presence of a vehicle. By monitoring the APD output every clock cycle, the presence of a vehicle in the detection zone can be determined along with the times associated with the vehicle entering and leaving the zone. The detection data are transmitted to a remote computer that displays or records the desired traffic information. For example, when a vehicle passes through the detection zone, the front of the vehicle first blocks laser beam 1 at time  $t_1$  and then blocks laser beam 2 at time  $t_2$ . Then, its rear will unblock laser beam 1 at  $t_3$  and laser beam 2 at  $t_4$ . Based on these time stamps, the front and rear velocities of the vehicle as well as the acceleration and length of the vehicle can be calculated [8].



**Figure 2-1. Working principles of the Laser-photodiode Based Detection System.**

The two linear laser beams that are used as laser sources are shown in Figure 2-2 (a), where the beams were imaged indoors through the use of a laser viewer. Figure 2-2 (b) shows the two laser beams projected across a lane of a highway when a vehicle is at the edge entering the field of view of the first laser/senor pair. The laser beam is reflected from the ground and is focused onto a 7.5 mm linear APD array through an optical system. The APD is an array of 25 photodiode elements, of which 24 are used. There are 24 signals for Laser1 and another 24 signals for Laser2. Each pair of element signals of Laser1 and Laser2 will turn out a set of vehicle parameters, giving a total of eight sets of vehicle parameters. The purpose of acquiring multiple vehicle parameters is to improve the reliability of the parameters and to acquire the outline profile of a vehicle.



**Figure 2-2. Laser-based detection system overview.**

When the Rabbit3200 reads the signal data, it packs data into TCP Packages and sends them to a remote computer through an Ethernet port. Using a wireless router, the LBDS is a fully un-tethered system allowing mobile systems to connect wirelessly to it in order to obtain the desired data. The vehicle parameters are calculated and displayed in a remote computer.

## 2.1 Architecture of the LBDS

The architecture of the LBDS is shown in Figure 2-3. It is composed of an opto-mechanical subsystem for laser-beam collection and adjustment, signal amplification and shaping circuitry to pick up the laser signals for conversion to digital signals, a Rabbit 3200 system for data acquisition, processing and data communication brokering, and a computer for parameter calculation and record/replay/reuse of data.

The opto-mechanical laser receiver collects the laser signal reflected from the ground into the APD array. The APD array generates signals when the laser beam is present. The signals are then amplified by the signal amplification and shaping circuitry. The opto-mechanical laser receiver has a very narrow field of view. When the system is moved to different heights, the laser sources must be adjusted so that the projected linear laser beam stays within the systems field of view. The opto-mechanical system is very sensitive to the signal output from the APD that is mounted to it. Our system should be adjusted accordingly relative to the height of the detector from the ground.

The signal amplification and shaping circuitry is used to pick up the signals generated from the APD array, and then amplify and shape them. It outputs digital signals that are read by the Rabbit 3200 core module. Considering that 48 signals will be read in the future instead of the 16 signals that are currently read, we use bus technology to connect the Rabbit core module to the signal amplification and shaping circuitry.

The Rabbit acts as a PWM generator to trigger the laser diode modules at 10 kHz. The laser diode module produces a linear laser beam with a full fan angle of 15 degrees that is projected towards the ground. Some of the reflected laser beam from the ground will fall onto the front of the receiving lens of the opto-mechanical system, which then falls onto the APD array causing it to output a pulse signal. The Rabbit core is triggered at the same frequency as the laser diodes to read the digital signals coming from the signal amplification and shaping circuitry. Every 10ms, the Rabbit module sends one TCP package containing 100 sets of data to a computer over a network.

The computer calculates the parameters of vehicles as they pass underneath the detector, and then displays, records, or replays the results.

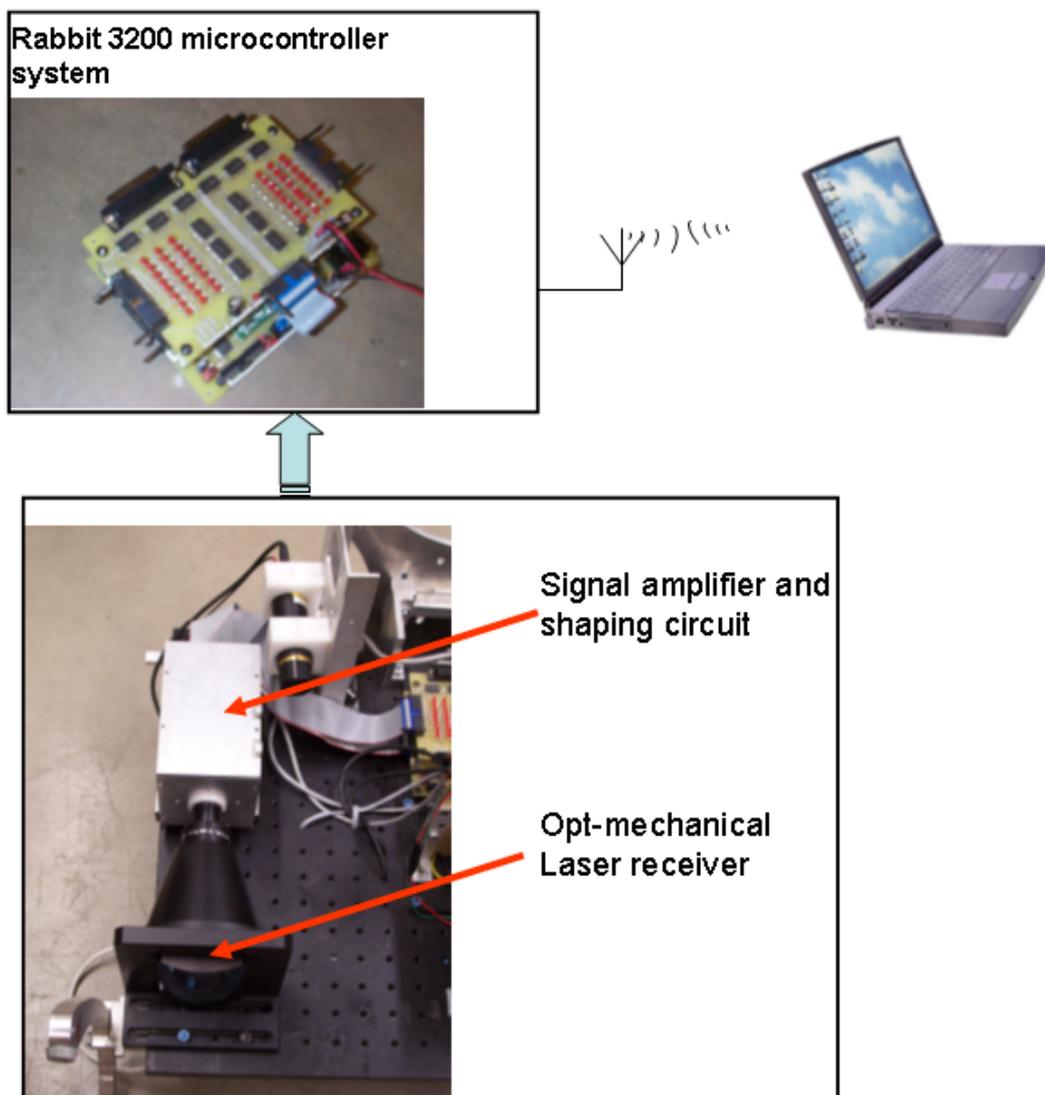


Figure 2-3 The architecture of the LBDS

## 2.2 Opto-Mechanical System

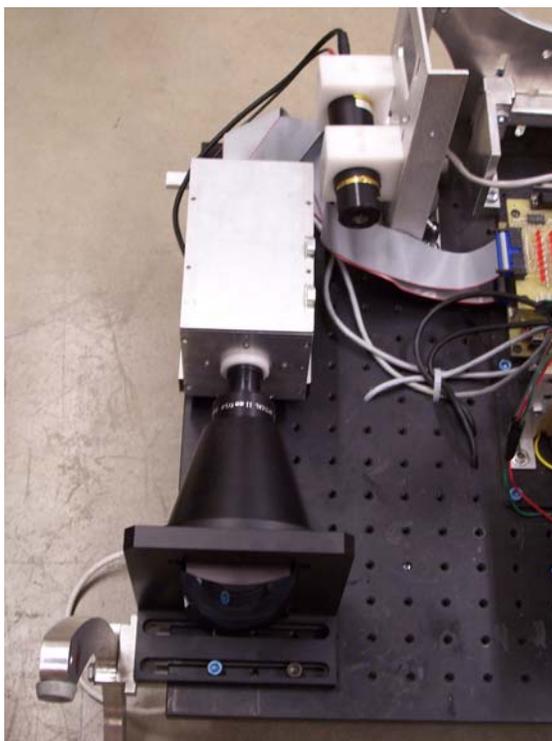
The optical system of the LBDS is one of the most important subsystems since it directly and adversely affects the overall performance of the LBDS.

### 2.2.1 New Structure of the Optics System

A new optical system has been designed in conjunction with JML Optical Industries, inc. The initial prototype is shown in Figure 2-4.



**Figure 2-4 Prototype of the new optical system.**



**Figure 2-5 Assembled optics-sensor pair.**

### **2.2.2 New Laser Module**

The laser module we used in the previous design could only provide a laser line 1.5m in length at the road surface 8m far away from the module. This is not long enough to cover the lane width of 3.8m of a highway. Two laser modules were therefore used on one side to combine the two laser lines into one line, hoping to solve the problem. But the combined laser line was only 2.5m long since an overlapping area is needed for the combination. So the problem still existed. In addition, it was difficult to position the two laser lines in one straight line.

To enable acquisition of a new laser module that meets our requirements, a laser module producing company developed a laser module for our application. The laser module has 60W peak power with a pulse frequency up to 10kHz. The length of the laser line on a road surface 8m away from the module can be longer than 4m, and the intensity distribution along the laser line is close to uniform. The performance of the new laser module is helpful to attaining correct vehicle detection. Figure 2-6 shows the comparison between the new laser line and the old laser line. The upper line is the new laser, and the lower line is the old laser which is combined by two laser lines.

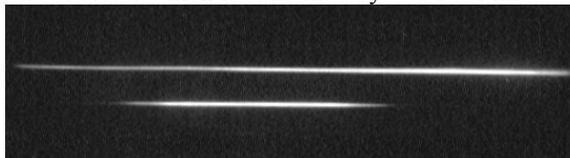


Figure 2-6. Laser lines

The second batch sample of laser modules is from Power Technology Inc. However, they exhibit overheating issues cannot be used for extended LBDS testing. The new laser modules operate at 24 volts as compared to the old models at 12 volts. A new power supply circuit was designed and fabricated to provide the necessary 24 volts. Figure 2-7 shows the laser module profile.

In order to do a comparison of the new laser modules with the older ones, outdoor testing, roof testing and highway testing were conducted. New laser module mounts were fabricated for the tests. Testing results showed that the new lasers are brighter, benefiting signal acquisition due to an increase in signal strength. However, electrical noise from the powerful drivers of the new laser modules have a strong effect on the front-end circuits and result in increased noise compared to the previous setup. Opto-electrical coupling components were added to take steps in avoiding the noise from new laser modules. The design is illustrated in the next section.



Figure 2-7 Laser modules

### 2.2.3 Use of a Laser Pointer

The laser light we are now using in the LBDS is an invisible infrared laser. It is difficult to align the laser line to the correct position on roadway. To solve this problem, we use a pair of green laser pointers to help the alignment of the infrared laser. The laser pointers are mounted on the laser mounts for housing the infrared laser module. One laser pointer is used on one side. The laser pointer points the same direction as the infrared laser on the same mount points, and the spot of the laser pointer will be in the center of the infrared laser line on the road. In this way, the infrared laser can be aligned easily to the required detection area for correct vehicle detection.

## 2.3 Redesign of the Electronics System

### 2.3.1 Microcontroller Based Control System

Based on the working principle of the LBDS, the main tasks that the electronics system should implement include: triggering a pair of laser diodes at a determined high frequency; preprocessing the analog pulsed signal outputs from a pair of APDs and converting them to a digital signal that can represent the absence/presence of vehicle clearly; outputting the raw data to a remote computer which is tens of meters away from the LBDS for calculating traffic information. Moreover, some special technologies are applied to the electronics system to let it have high reliability and stability and strong anti-noise ability for actual applications. The block diagram of the system is shown in Figure 2-8.

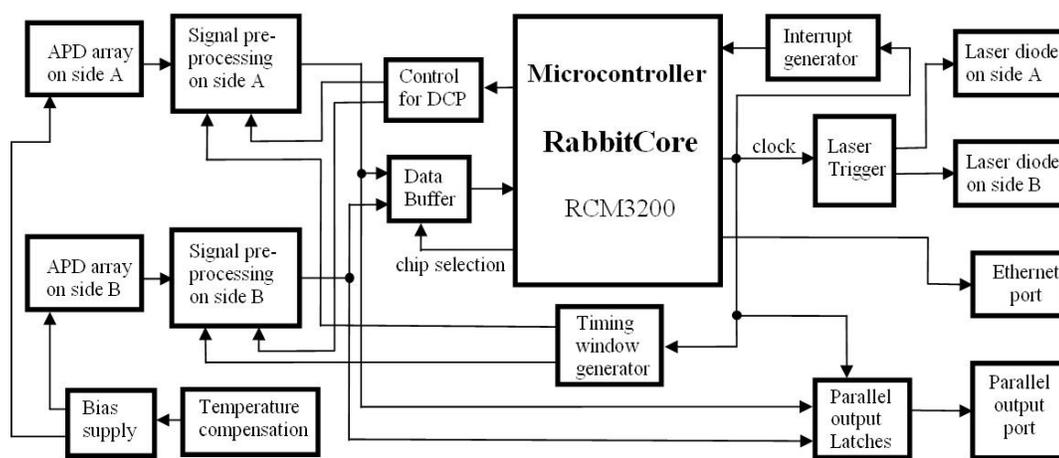
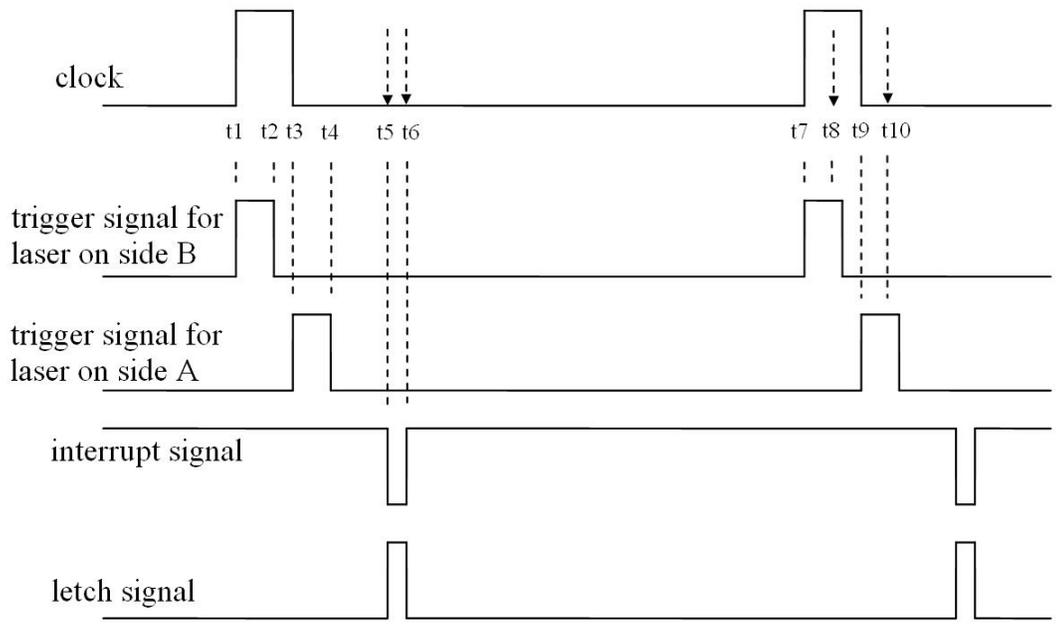


Figure 2-8. Block diagram of the electronics system.

The control core of the electronics system is a microcontroller “RabbitCore RCM3200 Module”. It generates a timing clock signal to synchronize the operation of all other parts of the system. The clock signal drives trigger circuitry to trigger the laser diodes on both sides separately and it also drives an interrupt generator to generate an interrupt application to request that the microcontroller readout all the data from both sides of signal pre-processing circuitries. The signal pre-processing circuits on side A and on side B are two similar circuitries that are used to amplify the signal output from relative APD array and convert it to binary digital data. The data buffer is used as bus-oriented receiver to get the data from signal pre-processing circuitry on both sides and transmit the data to the microcontroller through an 8-bit I/O port. There are two kinds of ports for detection data output, one of them is an Ethernet port that is already integrated onboard in the microcontroller, and the other one is a parallel output port. The Ethernet port is a 10/100Base-T Ethernet port through which the LBDS can connect to remote computers far away via TCP/IP communication, while the parallel output port enables the LBDS output the data almost no delay. Both kinds of outputs are available at the same time. The digitally controlled potentiometers (DCP) which are used in each channel of the signal preprocessing circuitries for optimal self-calibration are controlled by the microcontroller. The timing window is used to reduce the effect of noise on the detection data, and the temperature compensation is designed for high reliability of detection over a wide range of temperature.

The operation timing sequence of the electronics system is shown in Figure 2-9. The top line is the waveform of clock signal. Its frequency is 10.8 KHz and its duty cycle is 10%, so the period between  $t_1$  and  $t_7$  is  $92.5 \mu\text{s}$  and the period between  $t_1$  and  $t_3$  is  $9.25 \mu\text{s}$ . The second line and the third line are waveforms of trigger signals for laser diodes on side A and on side B. Since the laser diode module is triggered at the rising edge of the trigger pulse, the laser diode on side B will be

triggered at t1, and that on side A will be triggered at t3. The advantage of triggering the laser diodes on both sides at different times is that the crosstalk between side A and side B can be avoided easily by using timing windows. The fourth line is the waveform of interrupt signal, which will generate an interrupt application at t5, and the fifth line is the waveform of latch signal.



**Figure 2-9. Operation timing sequence of the LBDS.**

In every clock cycle, the laser diode on side B is triggered at t1 to project a narrow pulsed laser onto the road. This laser will then be reflected to APD array on side A (A2) if there is no vehicle to block the laser. Having received the reflected laser, A2 will output a narrow and small analog pulsed signal to the signal pre-processing circuitry on side A. This circuitry will amplify the signal and compare it with a reference voltage to convert the amplified analog signal to a binary digital signal, which will be kept until t8. Similarly, after the laser diode on side A is triggered at t3, the APD array on side B will be triggered at t3, the APD array on side B will output a pulsed signal to the signal pre-processing circuitry on side B. The circuitry will amplify the signal and convert it to a binary digital signal which will be kept until t10. After the binary digital signals of both side A and side B have been made ready to readout, an interrupt application to the microcontroller will be generated at t5 by the interrupt generator. Once the microcontroller responds to the interruption, it will readout all the binary digital signals on both sides through the data buffer before t7. Since the microcontroller has an Ethernet communication port integrated on board, the acquired and processed data will be transmitted to the remote computer through the port. At the same time when the interrupt application is generated, the latch signal is also generated for the parallel output port. The output of the parallel port will be refreshed at t6 by the binary digital signals from both signal pre-processing circuitries. By this way, the LBDS is able to transmit the detection data at a high speed. Figure 2-8 shows a prototype of the LBDS with the high performance electronics system.

**2.3.2 Laser Diode Module and the Trigger Circuit**

The laser source used in the LBDS is the ML series pulsed laser diode system, which is an integrated module. All components necessary for generating and projecting a laser line have been included in the module. Only a DC power supply and a TTL timing trigger signal are needed for its operation. The laser module will be triggered at the rising edge of the trigger signal to project a linear pulsed laser beam whose wavelength is typically 905nm and with a pulse width of 20ns.

The peak power of the new laser module is 60W - this is stronger than the old laser, which is 30 W. With the new laser, however, noise from the drive circuit is increased, which affects accuracy of the laser pulse signal acquisition. The optically coupled isolators are used to avoid the interferential signal from laser drive circuit.

The trigger circuitry is shown in Figure 2-10. The monostable multivibrators U3A and U3B are triggered at the rising edge and the falling edge of the clock signal respectively. When U3A or U3B is triggered, its “ $\overline{Q}$ ” pin will output a negative pulse with a pulse width of  $t_p = 0.45 \times R \times C$ , where  $t_p$  is the pulse width in ns,  $R = R_5 = R_6$  is the resistor in k $\Omega$ ,  $C = C_1 = C_2$  is the capacitor in pF. This negative pulse leads the transistor Q1 or Q2 to output a positive pulse which will trigger the laser module at the rising edge of the positive pulse. Since the trigger terminals of the laser diode modules have a low input resistance and the laser modules are triggered only at the rising edge of the triggering signal, we use power transistors Q1 and Q2 to have more power to drive the module, and we set  $t_p = 8\mu s$  to reduce the power consumption of laser modules. Another advantage of using a monostable multivibrator as the triggering signal generator is that the triggering of the laser module can be controlled. U3A and U3B can be enabled or disabled by pulling their “CLR” pins to high level or low level. If the monostable multivibrator is disabled, its output  $\overline{Q}$  will always stay high and the laser module will not be triggered whatever the clock signal changes. In our design, the laser can be switched on or off just by setting high or low level on “CLR” pins of U3A and U3B while the clock signal is still running. This function is necessary in the optimal self-calibration procedure where the laser is required to stop several times while the timing clock should be still running.

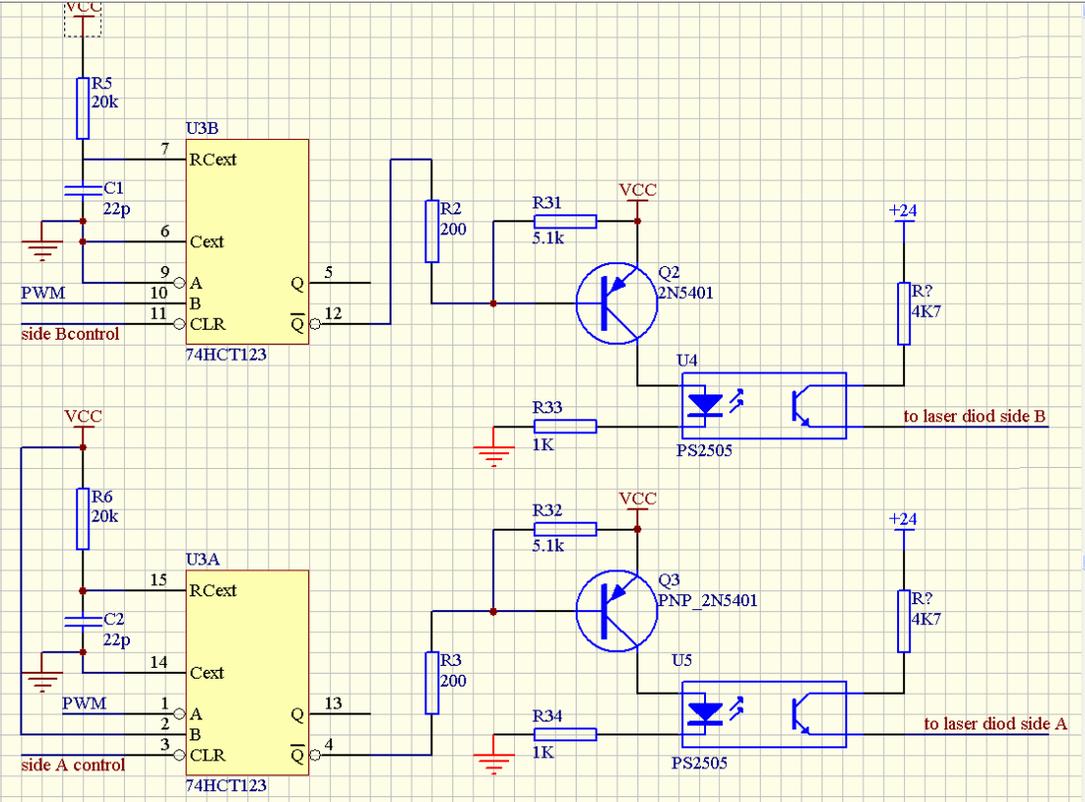
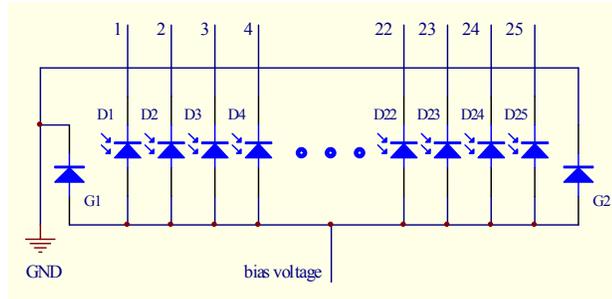


Figure 2-10. Laser diode modules trigger circuitry.

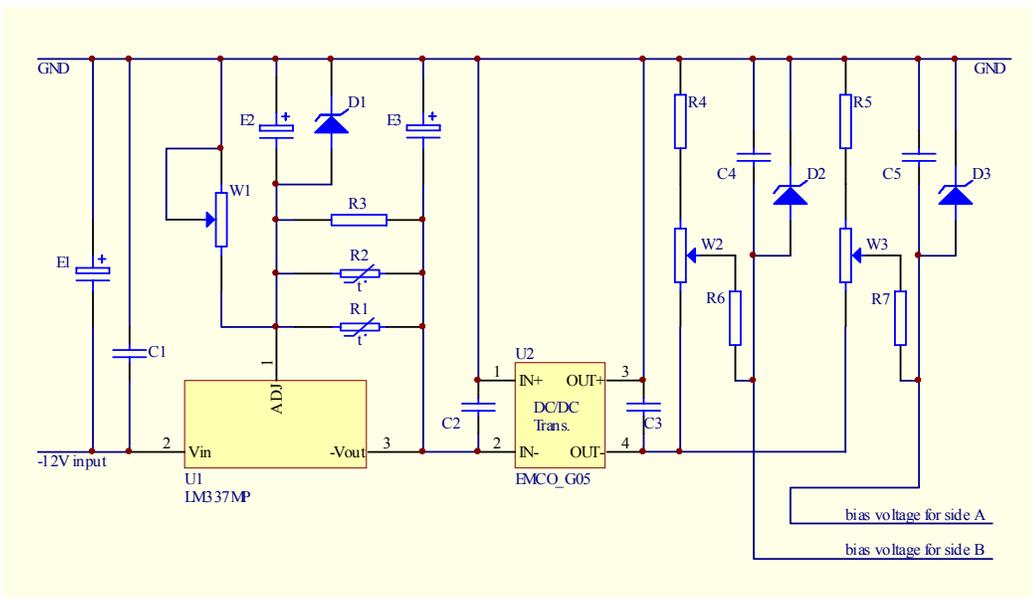
### 2.3.3 APD array and its bias voltage supply

The APD array used in the LBDS is a linear APD array that has 25 photodiodes located in a line. Figure 2-11 shows its internal connection. The anodes of APDs D1 to D25 are common connected and should be applied with a negative bias voltage ranging from -200V to -400V to let them work. G1 and G2 are guard ring diodes for protecting the chip. When the cathode of an APD is connected to signal pre-processing circuitry and the reflected laser falls on the APD, a small signal current will flow through the APD to the bias, which is the output signal of APD. The value of this signal will change as the laser intensity on the photodiode changes, and it will also change when the bias voltage changes. A stronger laser light on APD and/or a more negative bias voltage will lead to a bigger signal.



**Figure 2-11. Internal connections of the APD array.**

The bias voltage supply circuitry is shown in Figure 2-12. U2 is a DC/HVDC (High-Voltage DC) converter that can generate the bias voltage that we needed. When the input of U2 varies from -5V to -12V, the output of U2 varies from -150V to -460V. U1 is an adjustable voltage regulator whose output can be adjusted by W1, so that the output of U2 can reach an expected value. Since the optimal value of bias voltage for different APD arrays is different, two potentiometers W2 and W3 are used to adjust the bias voltage for both sides of APD arrays respectively. Two Zener diodes D2 and D3 with the breakdown voltage of 400V are used to limit the maximum negative value of the bias voltage. Thermistors R1 and R2 are used for temperature compensation. R1 and R2 are connected to the surface of APD array. When the temperature on APD changes, R1 and R2 change, then LM337MP voltage will change to control the output voltage of the DC/HVDC. R1 and R2 are negative thermistors, and R1 and R2 will be lower when temperature of APD is higher. The stability of bias voltage on both side A and B is very important for the signal acquisition - the signal is dependent on the bias voltage and amplifiers.



**Figure 2-12. Bias voltage supply with temperature compensation for the APD.**

**2.3.4 Signal Pre-Processing Circuitry**

The signal pre-processing circuitry plays an important role in the operation of the LBDS. It is used to amplify the detection signal output from APD and convert this amplified analog signal to a digital signal. The input signal comes from an APD sensor and its output is a binary logic signal that indicates the presence/absence of vehicles. Since the amplitude of the pulsed signal output from an APD is only a few  $\mu\text{A}$  and its width is about 100 ns, the circuitry is required to have large gain and wide bandwidth to convert the weak pulsed current signal from APD to a pulsed voltage signal with a comparable level and with a high signal/noise ratio. Considering that the signal from an APD has a heavy noise background (from sunlight and circuitry) and its amplitude fluctuates, the circuitry should have the ability to greatly reduce the effects from noise and amplitude fluctuation. Finally, the amplitude of the pulsed signal will vary as the environment changes, which may lead to incorrect vehicle detection. This situation may occur because the LBDS is to be used outdoors 24 hours a day and 7 days a week with a lifespan of several years. Therefore, the circuitry is also expected to have the ability to automatically adjust to an optimal state when the operating environment changes. As to the output of the circuitry, a binary logic signal is a better choice to represent the absence/presence of vehicles. This digital output can be easily realized by hardware to ensure high processing speed and reduce computational requirements for determining the absence/presence of vehicles, and it also significantly enhance the noise rejection ability of the system.

A schematic diagram of the circuitry for one element of an APD array is shown in Figure 2-13. The circuitry mainly consists of a three-stage preamplifier, a comparator, a monostable multivibrator and a digitally controlled potentiometer (DCP). Since a total of 48 elements of two APD arrays, 24 elements in each, are used for detection in the LBDS, there are 48 signal pre-processing circuitries similar to Figure 2-13 24 for one side.

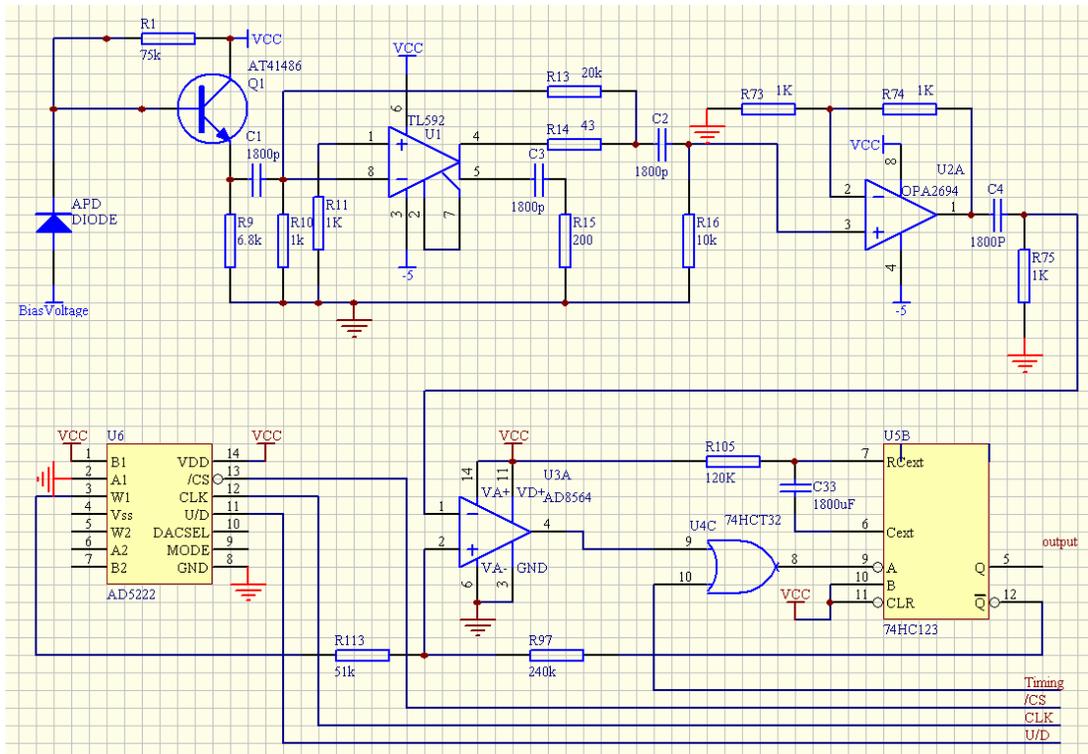


Figure 2-13. Signal preprocessing circuitry.

### 2.3.4.1 Three-stage Pre-amplifier

This pre-amplifier is used to convert the pulsed weak current signal output from an APD to a voltage signal and also to amplify this voltage signal to a comparable level. It is required to have large gain and a fast response speed. Therefore, this pre-amplifier is designed to be a three-stage preamplifier using two high quality operational amplifiers to make the signal amplification higher quality. The first stage is Q1 and the second stage is U1. All amplifiers offer low noise and high frequency performance over 150MHz. The current signal from an APD is amplified and converted to a considerable voltage signal by Q1, which will be coupled to U1 for a second amplification and then to the third U2A. The third amplifier has a potentiometer W1 in its feed back loop to adjust its gain manually, which can compensate the non-uniformity of the APD gain between the elements of an APD array.

In the previous design, this pre-amplifier used the combination of the AD8039 and AD8054. Although both of them have higher frequency performance, they are in smaller integration. But since there is a lot of noise from the background, they are not suitable to use in the pulse signal amplifier. According to the analysis of data acquisition, the stronger noise from sunshine is on the lower frequency from 0Hz to 5MHz, and the pulse signal that we used is on the higher frequency. The low filter has been used in the amplifier.

### 2.3.4.2 Comparator and Monostable Multivibrator

The comparator U2, which is a high speed comparator AD8564, is used to convert the analog signal to a binary logic signal by comparing the amplitude of the analog signal with a reference voltage that forms a threshold. Using a hardware comparator to convert the analog signal to a binary logic signal is much simpler and faster than using A/D conversion and comparison. The non-inverting input of U2

is connected to the reference voltage through resistor R11, and its inverting input is connected to the output of the preamplifier. If the level of the preamplifier output is higher or lower than the reference voltage, U2 will output either a low or high level to indicate the absence or presence of a vehicle in the detection zone. Since the output of the preamplifier is normally at a minimum level lower than the threshold, only the peak of the pulsed signal whose duration is about 200ns will be higher than the threshold. The output of the comparator will be a series of negative pulses with durations less than 200ns if there are pulsed signals in the preamplifier output. Conversely, the output of the comparator will stay at a high level if there are no pulsed signals in the preamplifier output.

Since the output of U2 is a narrow negative pulse when a pulsed signal occurs in the preamplifier output, it is difficult to sample such narrow pulses for detecting the absence or presence of a vehicle in the detection zone. To remedy this, a monostable multivibrator, component U3A in Figure 2-13, is used to keep the comparison result of U2 to the next cycle of the timing clock. In this way, the comparison result of U2 can be sampled easily. The output of U2 is coupled to the input of U3A by C4 and an “OR gate” U5A. The negative pulse output from U2 will trigger U3A to output a positive pulse from its output  $Q$  if U5A is controlled open by the low level of the timing signal. The pulse width of the output of U3A is determined by parameters of the external RC circuitry (R10 and C5) of U3A, which can be calculated as:  $t_w = 0.45 \times R_{10} \times C_5$ , where  $t_w$  is the pulse width in ns,  $R_{10}$  is the resistor in k $\Omega$  and  $C_5$  is the capacitor in pF. In the circuitry we set  $t_w = 110\mu s$ , which is slightly longer than the timing clock cycle  $T_{clock} = 95\mu s$ . If U3A is triggered continually in every  $T_{clock}$ , its output will always be at a high level. Once the trigger stops, the output of U3A will jump to a low level  $110\mu s$  after it was last triggered. Since the laser diodes are triggered at the beginning of every cycle of the timing clock and an APD will output a pulsed signal very near to the time it receives the reflected laser, the output of U3A will go to a low level about  $15\mu s$  after the beginning of the first timing clock cycle from when the trigger to U3A stops.

The output of U3A is related to vehicle detection. A high level means the laser beam is unblocked and a low level means the laser beam is blocked. Vehicle detection can be performed by sampling the output of U3A. This signal is sampled  $25\mu s$  after the beginning of every timing clock cycle to get correct and timely vehicle detection information.

### 2.3.4.3 Hysteresis to Overcome Noise and Signal Amplitude Fluctuation

The amplitude fluctuation of the preamplifier output will cause the output of U3A to toggle when the amplitude is close to the threshold, leading to incorrect vehicle detection. This problem often happens in the transition period when the laser beam is being blocked or unblocked, e.g., when a vehicle is moving into the detection area or leaving the area, the laser light can be blocked or unblocked gradually so the amplitude of the pulsed signal in the preamplifier output will go downward or go upward gradually through the threshold during the transition period. Using hysteresis on the comparator is a simple and effective way to solve this problem. With hysteresis, the comparator has two thresholds, an upper threshold and a lower threshold. When the peak-to-peak value of the amplitude fluctuation is in the range between the upper and lower thresholds, the effects of fluctuations can be eliminated.

Resistor R12 is used to configure comparator U2 with hysteresis. It connects the complementary output  $\bar{Q}$  of monostable multivibrator U3A and the non-inverting input of comparator U2 to form a positive feedback. The output  $\bar{Q}$  of U3A will then contribute to switch the threshold of U2 together with the reference voltage. The voltage on the non-inverting input of U2 can be calculated as:  $V_a = \frac{R_{12}}{R_{11} + R_{12}} V_{ref} + \frac{R_{11}}{R_{11} + R_{12}} V_{\bar{Q}}$ , where  $V_{ref}$  is the wiper output of U4, and  $V_{\bar{Q}}$  is the output  $\bar{Q}$  of U3A, which can be either 0V or 5V.

When monostable multivibrator U3A is triggered, its output  $\overline{Q}$  will go to a low level that will be kept to the next timing clock cycle. This status of  $\overline{Q}$  through R12 will drive the level of the non-inverting input of U2 lower than when R12 is not connected. This level is “low threshold”. As long as the amplitudes of the sequential pulsed signals are higher than the “low threshold”, U2 will output negative pulses reliably and keep the output  $\overline{Q}$  of U3A at low level. However, if the amplitude of the pulsed signal is lower than the “low threshold”, U2 will not output negative pulses any longer, so the output  $\overline{Q}$  of U3A will jump to a high level  $15\mu s$  after the beginning of the timing clock cycle, which will lead to a higher level on the non-inverting input of U2. This higher level is “high threshold”. During the next clock cycle, only when the amplitude of the pulsed signal is higher than the “high threshold”, can U2 output a negative pulse to trigger U3A and force the output  $\overline{Q}$  of U3A to a low level, which will drag the non-inverting input of U2 to “low threshold” again. If this condition happens, U2 will output negative pulses over the sequential clock cycles until the signal amplitude becomes lower than “low threshold”. In this way, the effect of the amplitude fluctuation of the pulsed signal is greatly reduced.

#### 2.3.4.4 Digitally Controlled Potentiometer

Correctly setting the threshold of the comparator, i.e., “calibration”, is important to the proper operation of the circuitry because the amplitude of the pulsed signal output from the preamplifier will vary as the operation environment changes. To obtain correct detection results in different environments, the circuitry should have the ability to automatically adjust the threshold to proper values.

Component U4 in Figure 2-13 is a nonvolatile digitally controlled potentiometer (DCP), which is used to adjust the variable reference voltage for the threshold of the comparator to realize optimal self-calibration. Its wiper can be moved in 128 steps between its two terminals under the control of 3-wire digital signals  $\overline{CS}$ ,  $\overline{INC}$  and  $U/D$ . When the wiper is moved between the bottom and the top terminals of the internal resistor array, the wiper output will vary between 0V and 5V. Since the Rabbit microprocessor is used in the LBDS, the variable reference voltage can be adjusted by the DCP automatically under the control of the microprocessor.

#### 2.3.4.5 Timing Window

A timing window is an efficient method to reduce disturbances of external linear noise. It cannot reduce the random noise such as sunlight. Only signals within the timing window can pass while all signals (e.g., noise) outside the window will be cut off. The timing window is important to these electronics. Since the monostable multivibrator U3A will be triggered as long as a negative pulse is exerted on its input pin “A”, it is easily disturbed. Component U5A is an “OR gate” that is used to set up the timing window. It is inserted in the connection between the output of U2 and the input pin “A” of U3A, and acts as a strobe to let only the real trigger pulses pass. One of its inputs is connected to a “timing” control signal whose waveform is as the upper line in Figure 2-14. The “timing” signal will fall to a low level when the laser diode is triggered and rise to a high level about  $4\mu s$  after. In the period when “timing” is at a low level, U5A is opened and the negative pulse from U2 can pass through U5A to trigger U3A. This period is the “timing window”. However, when “timing” is at a high level, no signal can pass through U5A, so U3A is isolated from noise and disturbances. The lower line in Figure 2-14 is the pulsed signal in the preamplifier output. The “timing” signal is generated by a monostable multivibrator, which is triggered by the timing clock. The width of the “timing window” can be adjusted by changing the external RC parameters of the monostable multivibrator.

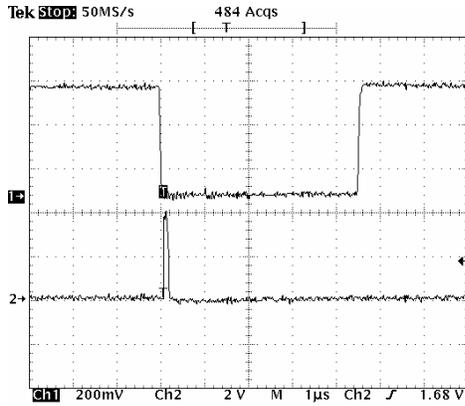


Figure 2-14. Timing window.

### 2.3.4.6 Pre-board Noise Data Analysis and Data Processing

The pre-board is in the front of acquisition system, and it is important to the system as sunlight and oscillating circuit noise effect signal acquisition. To provide the signal oscillating spectrum in Figure 2-15, signal analysis experiments were carried out using a spectrum analyzer. It was found that a high frequency range provided the best S/N ratio. The sunlight noise signal is on the lower frequency range, the pulse signal is on the higher frequency range, we used the high-pass filter to improve the S/N ratio. The old board that used the two-stage amplifier, the filter frequency is not high enough that sunlight noise affects the pulse signal. If the frequency is higher, the signal of the second stage amplifier is very weak. The signal acquisition circuitry was redesigned and adjusted to reduce the optical noise due to sunlight and oscillating noise.

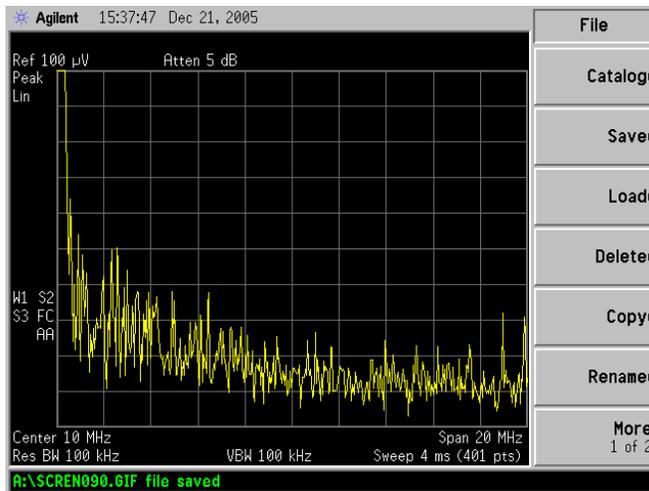
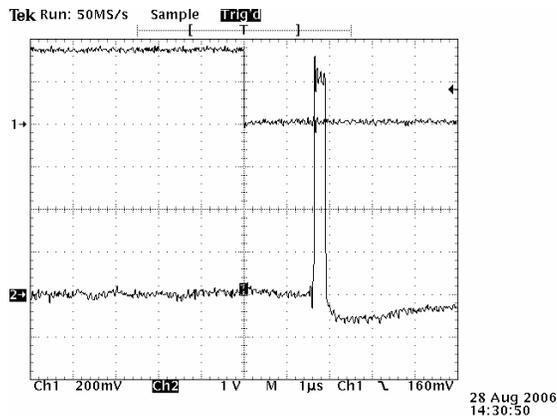
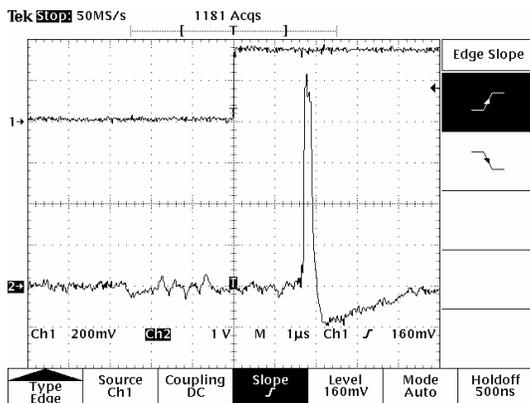


Figure 2-15 Pre-board sunlight signal spectrum



**Figure 2-16 Outside testing signal on time domain.**



**Figure 2-17 Roof testing signal on time domain.**

### 2.3.4.7 Arrangement of Printed Circuit Boards in Signal Box

The new system uses 24 detection channels on one side, i.e., 24 circuits same as the circuit shown in Figure 2-13 should be installed in a signal box. To make the signal box have a smaller size that fits the optical system, the arrangement of all the components and the PCB boards should be carefully designed. At present, we use large scale integrated circuits to reduce the required space of PCB board, so that 8 circuits for 8 channels can be designed in one small sized board. 3 such boards make 24 channels for detection in one side. These 3 boards are installed in the signal box horizontally as 3 layers with one end connecting the APD board and the other end connecting a back board which has some connectors to connect the main control board by cables.

### 2.3.5 Power Supply Circuit and DC/DC Converter

The power supply circuit provides various DC powers needed by the electronics system of the LBDS. Its input is 24VDC, which comes from a powerful 24VDC power supply, and its outputs are +5V, -5V, and -12V, as shown in Figure 2-18. With the VKP60xP 60 Watt triple output half brick DC/DC converter, the 12VDC input is converted to +5V, +12V and -12V DC power output. The +5V is used by the main control system, and the +24V is used to drive the laser diode module. Two voltage regulators U2 and U3 are used to obtain -5V from -12V supply. The +5V and -5V are used to power the signal pre-processing circuitries on both sides. The -12V is also used to power the negative bias voltage circuitry for the APDs. In the previous design of the LBDS, the power supply was a DC/DC converter which converts the 24DC to +5V, -5V, +12V, and -12VDC. The reason of using the 24VDC as the input of the power supply is that the LBDS can be made mobile with the trolley on the truss, and the DC power supply can be easily obtained from the rails of the truss.

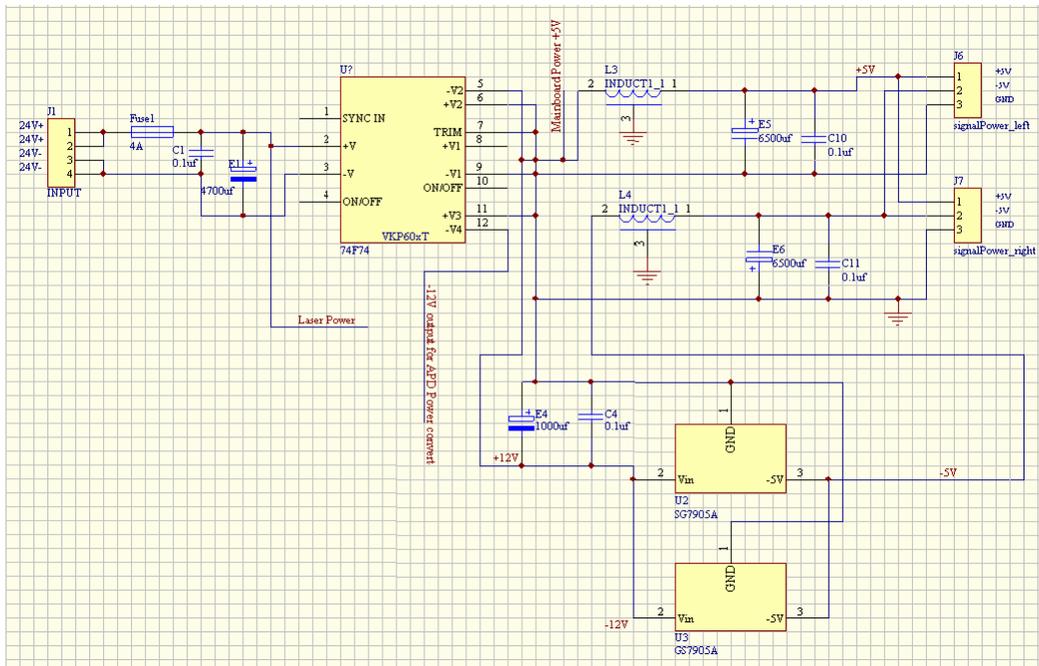


Figure 2-18. Power supply DC/DC converter circuitry.

### 2.3.6 Temperature Compensation for Responsivity of APD

The responsivity of an APD will change as the temperature of the operating environment changes due to its temperature performance [15], which results in the fact that the signal amplitude of APD output and accordingly the amplitude of pre-amplified signal will be different in different temperatures even though the other detection conditions remain the same. The higher the environment temperature, the smaller the signal amplitude of the APD. Figure 2-19 shows the relationship between the amplitude of the pre-amplified signal and the environment temperature. This temperature performance of the APD will lead to misdetection when the LBDS works in a wide range of temperature, since the LBDS detects vehicles just by comparing the amplitude of pre-amplified signal with a threshold. If the high temperature makes the signal amplitude become so small that it is always lower than the threshold even if no vehicle blocks the laser, the signal processing circuitry of the LBDS will not be able to distinguish whether there is a vehicle in the detection zone or not. Considering that the LBDS is designed to be used outdoors and the highest outdoor temperature in hot summer day may exceed  $40^{\circ}\text{C}$ , temperature compensation is necessary for correct operation of the LBDS.

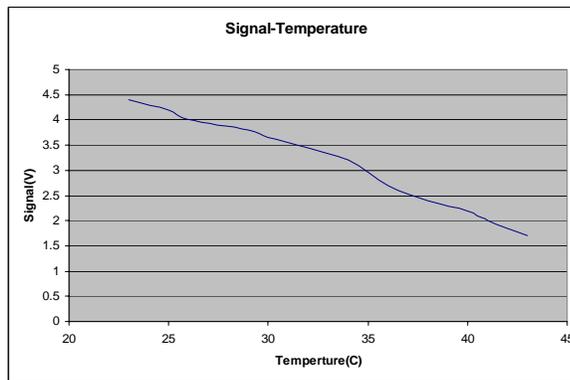
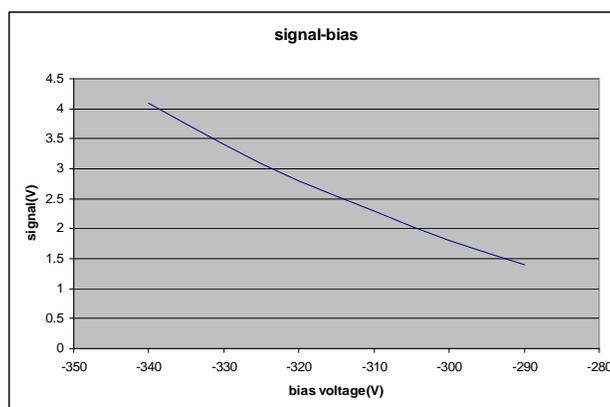


Figure 2-19. Relationship between signal and temperature.

Temperature compensation can be realized by changing the bias voltage of the APD. The responsivity of an APD will change as the bias voltage changes according to its performance. A more negative bias voltage will lead to bigger signal amplitudes. Figure 2-20 shows the relationship between the amplitude of the pre-amplified signal and the bias voltage, which was measured at a temperature of 25°C. To reduce the effect of temperature variation on the amplitude of the signal, the bias voltage should be adjusted to be more negative when the temperature rises higher. Although it seems possible that a big amplitude of signal in a wide temperature could be obtained only by setting the bias voltage at a high negative value, it is not realizable. The bias voltage of APD has a highest negative voltage limit which is different in different temperatures. If the bias voltage exceeds this limitation, the signal will vibrate and cannot be used for detection.

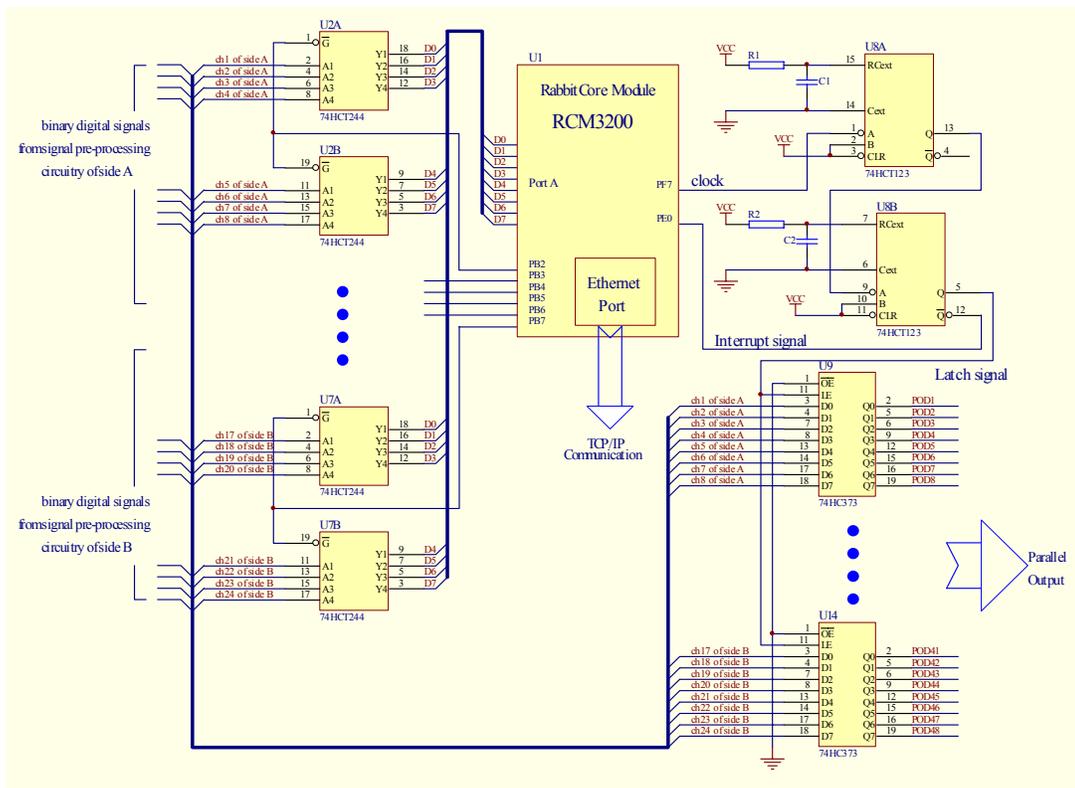
The circuitry for temperature compensation is shown in Figure 2-12. It is a temperature controlled variable bias voltage supply. R1 and R2 are thermistors with negative temperature coefficients. They are mounted on the covers of the APD arrays and work as temperature sensors. When the temperature rises, the resistances of R1 and R2 reduce, which lead to voltage increments on the outputs of U1 and U2. The bias voltages on APD arrays are therefore raised. In this case, the signal decreasing tendency caused by the higher temperature will be counteracted by the signal increasing tendency caused by the more negative bias voltage. The result of a temperature compensation test of this circuitry showed that the signal amplitude can be kept at 3V over a wide range of temperature from 14°C to 42°C.



**Figure 2-20. Relationship between signal and bias voltage.**

### 2.3.7 Data Acquisition and Output

Data acquisition is the operation where the Rabbit microprocessor obtains the detection data from the signal pre-processing circuitry on both sides of the LBDS, and the data output is the operation where the detection data are transmitted from the LBDS to the remote PC for calculating the traffic parameters. To make the LBDS more flexible for the requirements of different applications, two kinds of data outputs are designed in the LBDS. One of them is parallel data output through a parallel port, and the other is serial data output by Ethernet communication through an onboard Ethernet port. The circuitry is shown in Figure 2-21.



**Figure 2-21. Data acquisition and output circuitry.**

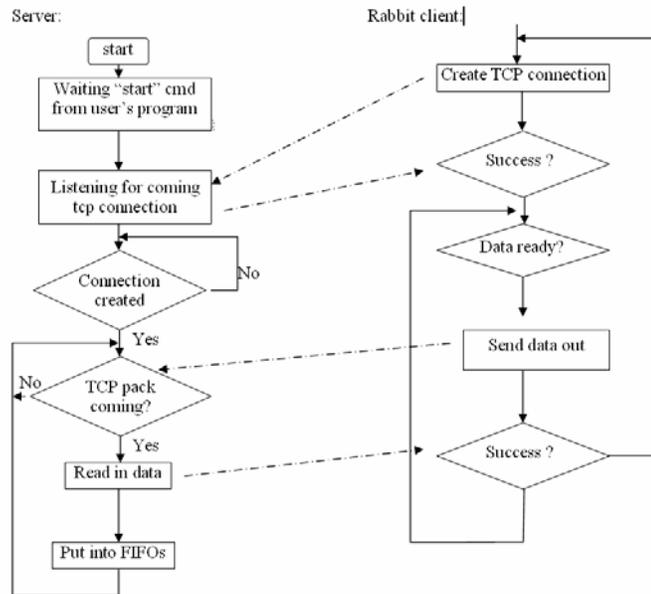
### 2.3.7.1 Rabbit Data Acquisition

The output signal from one channel of the signal preprocessing circuitry is a binary logic signal. 24 binary signals for 24 channels in one side combine 3 bytes of 8-bit data. The total 6 bytes of 8-bit data from both sides are fed to 6 data buffer chips U2, U3, U4, U5, U6, and U7. U8A and U8B make the interrupt generator and latch generator. When the microprocessor responds the interrupt application in every clock cycle, it reads out all 6 bytes data from both sides through the data buffers. The advantage of acquiring data in this way is that only one 8-bit input port in the Rabbit microprocessor is needed for reading out 6 bytes data, which reduces the numbers of the I/O ports needed for data acquisition and allows those I/O ports resources to be used for other purposes.

### 2.3.7.2 Output Data Through Ethernet Communication

Since the microcontroller RabbitCore RCM3200 module has a 10/100Base-T Ethernet port available on-board, the detection data can be transmitted via this port using TCP/IP communication. The network architecture for Ethernet communication in our application is client/server. The remote PC works as a server and the RabbitCore microcontroller in the LBDS works as a client. Both of them are assigned an IP address respectively. When the LBDS starts operation, the RabbitCore microcontroller will configure the Ethernet port and initialize the TCP/IP interface at first, and then it tries to set up the connection with the remote PC by sending a TCP segment to the PC. If the remote PC responds, the microcontroller sends back the acknowledgement. Then, the connection between the PC and the LBDS has been established, and the TCP socket is therefore opened. In the remote computer side, after it runs our application program “Xvehicle”, the server will create a TCP socket and wait for a connection from the LBDS by listening for the connection from the Rabbit module on the TCP port “2002”. If the connection is established successfully through a server-client handshake, the server will read the data coming from the TCP session and send the data to the user program through FIFOs. The flow chart of the server is shown in Figure 2-22. On the other hand, in the vehicle detection routine,

the microcontroller stores the detection data that are obtained from the signal processing circuitry on both sides into one of two data arrays in memory in every clock cycle. When the data array being used is full, all data in this array will be put into a packet and the microcontroller will start to send the packet to the remote PC via TCP/IP communication, while the storage of the detection data will be switched to the other data array. Both data arrays are used alternately, so the detection data can be acquired every clock cycle and sent to the remote PC in packets continually. Meanwhile, when the Rabbit module sends out a package, a tcp\_tick (&sock) should be called. Otherwise, the TCP package sending buffer will be full and the data package will not be sent out. With Ethernet communication, the operator's commands for some actions of the LBDS can also be sent from the remote PC to the LBDS to remotely control the LBDS.



**Figure 2-22. The flow chart of TCP/IP communication.**

### 2.3.7.3 Output Data Through Wireless Ethernet

Besides the cable Ethernet communication with the remote computer, the LBDS has the ability to communicate by wireless Ethernet communication. A wireless router is used in this case. The Ethernet port is connected to the router which is installed near the LBDS by a cable. The remote computer can receive the detection data through the wireless router by wireless Ethernet communication if it has a wireless networking capability. Utilizing wireless network technology is more favorable since the LBDS may be mounted above highway and the desired mobility of LBDS requires that communication with the remote computer be wireless.

### 2.3.7.4 Parallel Port Output

Unlike the Ethernet communication, the parallel port outputs the detection data every clock cycle. The six 8-bit detection data from both sides are connected directly to the inputs of the latches of parallel ports U9, U10, U11, U12, U13 and U14. After the six 8-bit data from both sides are ready to be readout in every clock cycle, a latch signal is generated to let the six latches U9 – U14 refresh their outputs. The parallel output is completely controlled by the peripheral circuitry, it will not occupy any CPU time of the microcontroller. These 48 digital signals can be transmitted to a remote computer through a DIO-96 card which is installed in the computer to readout the data for calculating traffic parameters.

### 2.3.8 Optimal Self-Calibration

The amplitude of the signal output from a two-stage preamplifier will be affected by the operating environment, e.g., sunlight, weather and road surface conditions, and the noise magnitude will also vary as the environment changes. A threshold that is well configured for a particular environment may not be suitable for another. If the threshold is fixed, incorrect detection may happen when the environment changes. The reliability and the correctness of the detection result cannot be guaranteed. As a result, automatically and periodically setting the threshold of the comparator to a proper value, i.e., “calibration”, is important to the proper operation of the LBDS.

Based on this consideration, a calibration method is used to make the detection more reliable and flexible. The idea of this calibration is to set the threshold of the comparator U2 at the midpoint between the lowest maximum amplitude of the pulsed signals and the highest amplitude of noise disturbances, which will lead to stronger ability for laser signal detection and noise resistance. The calibration can be done by adjusting the wiper position of the DCP, which is controlled by a microcontroller. If a single threshold is used, the calibration is relatively simple. But after hysteresis is employed in the circuitry, two thresholds will be in action. In this case, calibration is more complicated.

Figure 2-23 depicts equivalent circuitry of components U2 and U3A in Figure 2-13. This is a comparator with hysteresis. The threshold  $V_a$  can be calculated as follows:

$$V_a = \frac{R_2}{R_1 + R_2} V_{ref} + \frac{R_1}{R_1 + R_2} V_{out}$$

Assuming  $V_{out}$  can only be either 0V or 5V, we then obtain:

$$V_a^- = \frac{R_2}{R_1 + R_2} V_{ref} \quad (V_{out} = 0V)$$

$$\text{or, } V_a^+ = \frac{R_2}{R_1 + R_2} V_{ref} + \frac{R_1}{R_1 + R_2} \cdot 5V \quad (V_{out} = 5V)$$

Because of the hysteresis, the  $V_{out}$  level will change not only due to the level of  $V_{in}$  but also due to the previous level of  $V_{out}$  even when  $V_{ref}$  remains unchanged, i.e., if  $V_{out} = 5V$ , only when  $V_{in} = V_{in}^+ > V_a^+$  will  $V_{out}$  jump to 0V, however, if  $V_{out} = 0V$ , only when  $V_{in} = V_{in}^- < V_a^-$ , will  $V_{out}$  jump to 5V. As a result, the comparator with hysteresis can effectively avoid toggles on  $V_{out}$  when  $V_{in}$  is close to one of the two thresholds  $V_a^+$  and  $V_a^-$ .

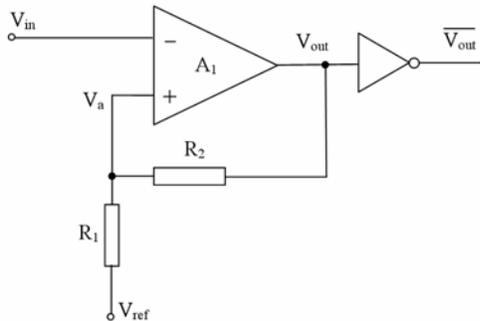


Figure 2-23. Comparator with hysteresis.

The idea of “calibration” here is to set  $V_{ref}$  to a value where the amplitude margins of the signal and the noise are the same. The principle of the calibration is described in Figure 2-24. Lines ① and ② are the relations between  $V_{ref}$  and  $V_a$ , while line ③ is  $V_a = V_{ref}$  without feedback. Suppose  $V_{signal}$  is the peak value of the pulsed signal and  $V_{noise}$  is the peak value of noise. Their relative values of  $V_{ref}$  are  $V_H$  and  $V_L$  respectively. We want to find the optimal  $V_{ref}$  value  $V_M$  that makes the length between points  $a$  and  $V_{amH}$  the same as the length between  $b$  and  $V_{amL}$ . These two lengths are amplitude margins of the signal and the noise. Here,  $V_{amL}$  is the level that  $V_{out}$  will remain at (0V) if the noise level is higher than this level, and  $V_{amH}$  is the level that  $V_{out}$  will remain at (5V) if the signal level is lower than this level. The value of  $V_M$  can be derived by letting the length between  $a$  and  $V_{amH}$  equal to the length between  $b$  and  $V_{amL}$ . The calculation is as follows:

$$V_{amL} = \frac{R_2}{R_1 + R_2} \cdot V_M, \quad V_{amH} = \frac{R_2}{R_1 + R_2} \cdot V_M + \frac{R_1}{R_1 + R_2} \cdot 5,$$

$$V_{noise} = \frac{R_2}{R_1 + R_2} \cdot V_L, \quad V_{signal} = \frac{R_2}{R_1 + R_2} \cdot V_H + \frac{R_1}{R_1 + R_2} \cdot 5,$$

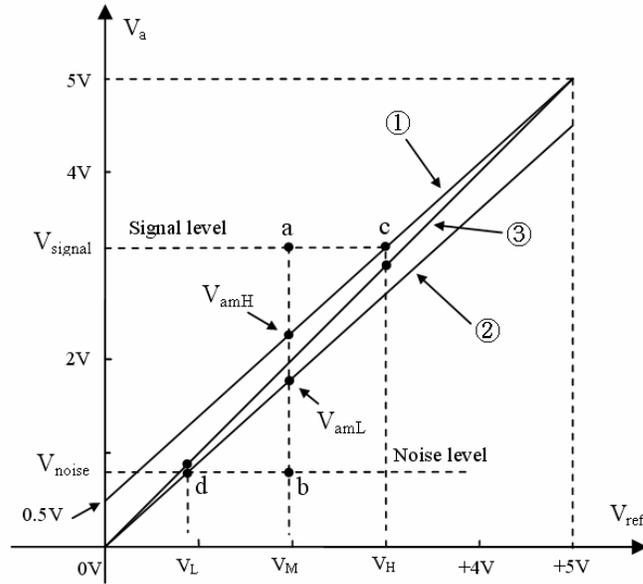
$$\text{Let: } V_{amL} - V_{noise} = V_{signal} - V_{amH}.$$

Then we obtain:

$$\frac{R_2}{R_1 + R_2} \cdot V_M - \frac{R_2}{R_1 + R_2} \cdot V_L = \frac{R_2}{R_1 + R_2} \cdot V_H + \frac{R_1}{R_1 + R_2} \cdot 5 - \frac{R_2}{R_1 + R_2} \cdot V_M - \frac{R_1}{R_1 + R_2} \cdot 5$$

$$\text{That is } \frac{R_2}{R_1 + R_2} \cdot (V_H - V_M) = \frac{R_2}{R_1 + R_2} \cdot (V_M - V_L), \text{ so we have: } V_M = \frac{1}{2} \cdot (V_H + V_L).$$

If  $V_H$  and  $V_L$  are determined,  $V_M$  can be obtained. But unfortunately  $V_H$  and  $V_L$  are unknowns and they need to be measured by the LBDS itself. Since there are no A/D converters available in the LBDS, the  $V_H$  and  $V_L$  values have to be determined by moving the wiper of the DCP (U4) between its two terminals, which will adjust  $V_{ref}$  between 0V and 5V to search for the wiper position where a transition on the output level of U3A occurs. The value of  $V_{ref}$  can be determined from the DCP wiper position. The flowchart of the calibration procedure is shown in Figure 2-25.



$$\textcircled{1} \quad V_a = \frac{R_2}{R_1 + R_2} V_{ref} + \frac{R_1}{R_1 + R_2} \cdot 5V (V_{out} = 5V)$$

$$\textcircled{2} \quad V_a = \frac{R_2}{R_1 + R_2} V_{ref} (V_{out} = 0V)$$

$$\textcircled{3} \quad V_a = V_{ref} (NO \quad V_{out})$$

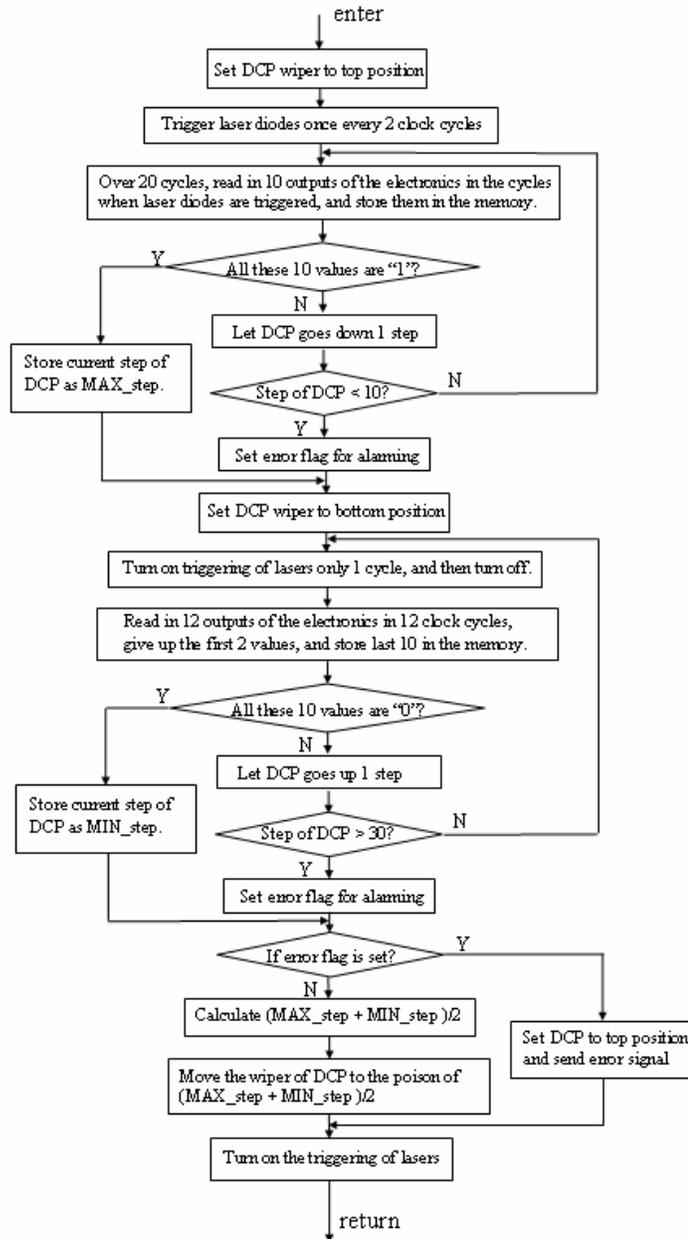
**Figure 2-24. Demonstration of calibration.**

Since the DCP wiper position is related to  $V_{ref}$ , the value of  $V_{ref}$  can be set by moving the wiper to certain positions. At the beginning of the flowchart, the laser diodes are triggered once every 2 clock cycles, and the wiper position of DCP is set to the highest point, where  $V_{ref} = 5V$ . At this time,  $V_{out} = 5V$  and  $\overline{V_{out}} = 0V$  since the signal pulse amplitude is lower than 5V. The microcontroller then controls the DCP to move its wiper downward step by step. The microcontroller tests  $\overline{V_{out}}$  in the clock cycles when laser diodes are triggered. 10 values of  $\overline{V_{out}}$  will be tested over 20 clock cycles every step. Testing 10  $\overline{V_{out}}$  values over 20 clock cycles is due to the requirement to eliminate the effect of amplitude fluctuation of the pulsed signal and to initialize  $V_{out}$  to 5V for every cycle in which  $\overline{V_{out}}$  value is tested. When the 10 values of  $\overline{V_{out}}$  tested over 20 clock cycles are all at high level, the current step of the wiper corresponds to  $V_H$ , and the microcontroller will record this step for later calculations. Once  $V_H$  has been obtained, the microcontroller will change to find the  $V_L$  value. At this point, the wiper of DCP will be set to the lowest position and then move upwards step by step while 10 values of  $\overline{V_{out}}$  will be monitored over 12 clock cycles at every step. If the 10 values of  $\overline{V_{out}}$  tested over 12 clock cycles are all at low level, the current step of the wiper corresponds to  $V_L$ . The microcontroller will also record this step for later calculations.

Since we want to ascertain the maximum level of noise,  $V_{noise}$ , the strategy for determining it is as follows. Set the DCP wiper to a certain step, initialize  $V_{out}$  to 0V by triggering laser diodes only one clock cycle at the beginning of the test in this step, stop the triggering of the laser diodes over the following 11 clock cycles so that there is only noise on the input of the comparator, and then sample 10 values of  $\overline{V_{out}}$  over last 10 clock cycles in this step. At the first wiper steps of the DCP,  $V_{ref}$  is small, so  $V_a^- = \frac{R_2}{R_1 + R_2} V_{ref}$  will be lower than  $V_{noise}$  and input noise will cause  $V_{out}$  to stay at 0V. But as the wiper of the DCP moves upwards step by step and  $V_{ref}$  increases,  $V_{out}$  will jump to 5V as soon as  $V_a^- = \frac{R_2}{R_1 + R_2} V_{ref}$  is greater than  $V_{noise}$ . By sampling  $\overline{V_{out}}$ , the step corresponding to  $V_L$  can be determined and recorded. To initialize  $V_{out}$  in this case, 12 clock cycles are used at every step. The laser diode will be triggered once in the first cycle to force  $V_{out}$  to 0V, but won't be triggered in the remaining (11) cycles. The values of  $\overline{V_{out}}$  sampled in the first two cycles will be given up because the effect of the laser signal on the monostable multivibrator (U3A) may still exist, while the values sampled in later 10 cycles will be monitored for determining  $V_L$ . In this way,  $V_H$  and  $V_L$  can be obtained reliably.

After  $V_L$  and  $V_H$  have been obtained, the microcontroller can calculate  $V_M$  according to  $V_M = \frac{1}{2} \cdot (V_H + V_L)$  and get the corresponding step setting for the DCP. Once the optimal step setting is worked out, the microcontroller will first set the wiper of DCP to the bottom terminal, and then move the wiper to the expected position.

The calibration procedure has the ability to provide an error message if the operating environment is too difficult for the LBDS to operate correctly.

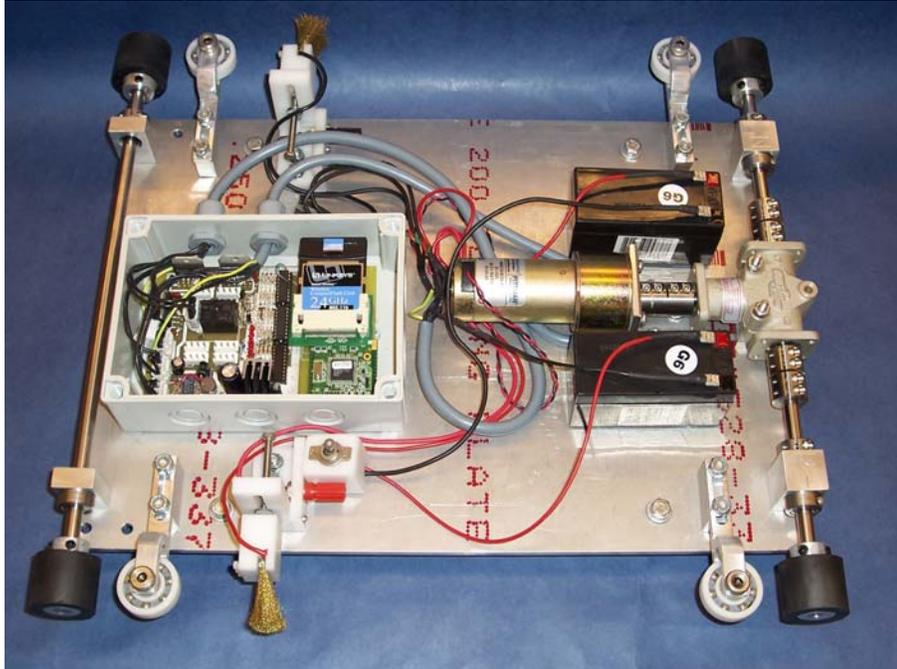


**Figure 2-25. Flowchart of calibration procedures.**

### 2.3.9 Remote Control for Trolley through Wireless Ethernet

The LBDS may sometimes be mounted on a trolley moving on a truss as a mobile LBDS. The motion of the trolley should be remotely controlled by the operator who stands on the ground to let the trolley move to the correct position. Radio remote control can be used for the remote control of the trolley, but since the wireless Ethernet communication is used in LBDS, it is favorable to utilize the wireless Ethernet communication for remote control of the trolley. To realize this idea, The Rabbit microprocessor is used for the control, and a new circuit board has been designed and fabricated as shown in Figure 2-26, and they communicate through the router. When the Rabbit microprocessor receives a command for the movement of the trolley through wireless Ethernet communication, it will transmit the corresponding control signals to the trolley, and also control the power supply of the

LBDS. In this way, the remote computer can be used not only to receive the detection data, but also to control the trolley.



**Figure 2-26 The trolley prototype**

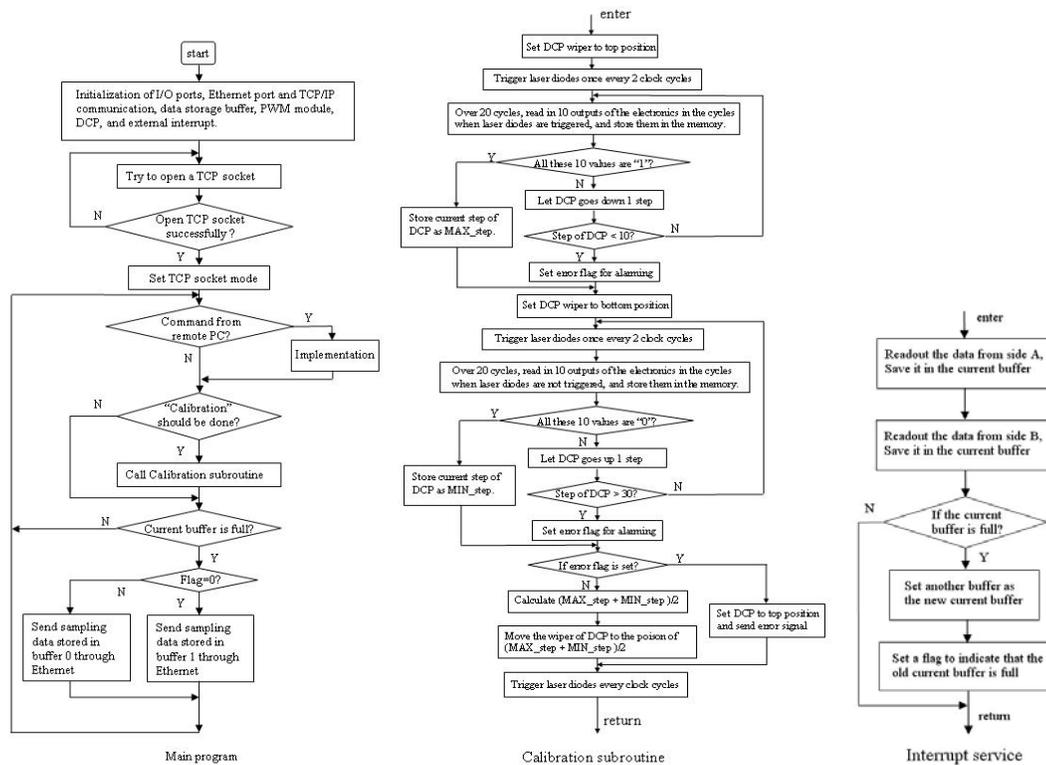
### **2.3.10 Software Design for Rabbit Microprocessor**

A Rabbit microprocessor is used in the LBDS, so a control program for the operation of the LBDS has been designed for the microprocessor. The program is written mainly in C language with some parts written in assembly language where the time constraints are strict. It consists mainly of a main program, a calibration subroutine and an interrupt service subroutine. The flowcharts are shown in Figure 2-27.

The main program is used to initialize all programmable components, send real-time data to the server and test whether re-calibration is necessary. After the system is powered on, the main program initializes the system and then goes into an infinite loop.

The interrupt service subroutine is used to read sampling data from signal pre-processing circuits. This subroutine will be carried out once every  $100\mu\text{s}$ . All 6 bytes, i.e., 48-bits, of sampling data will be readout at the beginning of the subroutine, and then the data will be put into a buffer for transmitting to a computer through an Ethernet cable. To ensure continuous data transfer, two buffers are used in turn to store and send data.

The calibration subroutine is used to set the wipers of all 48 DCPs to their optimal positions. The procedure of the subroutine has been described above.



**Figure 2-27. Flowcharts of control programming on the Rabbit microprocessor.**

*readData()*

This is an interrupt service subroutine. It reads in the 48-bit sampling data from port A and stores the data into two array buffers in turn.

*setXDCP ()*

This function is used to initialize all 48 DCPs when the system powered on.

*setPwm()*

This function sets the PWM module with required parameters to generate the timing clock signal.

*setPortInit()*

This function initializes all Rabbit ports that will be used in this system.

*calibration()*

This function is used to set all 48 DCPs to their optimal positions.

**2.4 New Design of the Mechanical System**

**2.4.1 Breadboard for optical system**

To facilitate optical alignment and mounting, an optical breadboard is now used as the base for the LBDS. All optical and opto-mechanical equipment are directly mounted to the optical base. The optical breadboard comes with an array of precisely spaced mounting holes. A half-inch optical breadboard was chosen for strength and stiffness. If desired, pockets can be milled out in order to

reduce the overall weight of the system. This step can be done once the exact positions of the optical and opto-mechanical subsystems are known.

### 2.4.2 LBDS Housing

In order to reduce potential damage to the LBDS due to weather conditions, a housing for the LBDS has been designed and fabricated out of plastic. The new housing prevents dust and rocks from damaging the optics and the lasers. However, new considerations must be taken in order to have a watertight housing.

### 2.4.3 New Mounting Mechanism for Highway Test

A new LBDS mounting mechanism for highway testing has been designed and fabricated. The new mounting mechanism allows for the rotation of the LBDS during field tests.. This enabled the alignment of the LBDS with the road curvature while the LBDS is positioned over traffic. The previous model required the LBDS to be pulled from its testing position, un-mounted and remounted in a slightly rotated position and then repositioned over the traffic. The new method reduces the amount of potential hazards from accidentally dropping equipment or tools onto the road surface and reduces the endangerment of human life.

## 2.5 Software on remote computer

With our application software “Xvehicle” running on it, the remote computer can receive detection data from the LBDS through wireless Ethernet communication or through parallel communication. The received raw data will be then used to calculate the traffic parameters such as velocity, acceleration and length of vehicles. These calculated parameters will be displayed on the remote computer. Figure 2-28 shows the snapshot of the screen of the remote computer.

File   Chart   OFF   Reset   Rabbit   Start   OneshotOff   Continue   AllSignals   Seperate Vel						
100 scans/10 msec (0.1000 msec/scan): displayed every 1 msec						
v1: 35.57 miles/h	v2: 33.96 miles/h		a: -2.841 m/(s <sup>2</sup> )	length: 3.929 n		
v1: 35.05 miles/h	v2: 34.25 miles/h		a: -1.398 m/(s <sup>2</sup> )	length: 3.955 n		
v1: 34.15 miles/h	v2: 35.57 miles/h		a: 2.453 m/(s <sup>2</sup> )	length: 4.015 n		
v1: 34.25 miles/h	v2: 35.57 miles/h		a: 2.269 m/(s <sup>2</sup> )	length: 4.049 n		
v1: 34.35 miles/h	v2: 35.25 miles/h		a: 1.560 m/(s <sup>2</sup> )	length: 4.041 n		
v1: 34.25 miles/h	v2: 35.15 miles/h		a: 1.547 m/(s <sup>2</sup> )	length: 4.039 n		
v1: 34.25 miles/h	v2: 35.15 miles/h		a: 1.542 m/(s <sup>2</sup> )	length: 4.051 n		
v1: 34.06 miles/h	v2: 35.67 miles/h		a: 2.755 m/(s <sup>2</sup> )	length: 4.088 n		
v1: 34.45 miles/h	v2: 35.36 miles/h		a: 1.552 m/(s <sup>2</sup> )	length: 4.096 n		
v1: 34.35 miles/h	v2: 35.36 miles/h		a: 1.719 m/(s <sup>2</sup> )	length: 4.092 n		
v1: 34.74 miles/h	v2: 35.46 miles/h		a: 1.219 m/(s <sup>2</sup> )	length: 4.127 n		
v1: 34.64 miles/h	v2: 35.57 miles/h		a: 1.567 m/(s <sup>2</sup> )	length: 4.128 n		
v1: 34.64 miles/h	v2: 35.46 miles/h		a: 1.389 m/(s <sup>2</sup> )	length: 4.123 n		
v1: 34.64 miles/h	v2: 35.46 miles/h		a: 1.389 m/(s <sup>2</sup> )	length: 4.123 n		
v1: 34.95 miles/h	v2: 35.57 miles/h		a: 1.054 m/(s <sup>2</sup> )	length: 4.145 n		
v1: 34.64 miles/h	v2: 35.57 miles/h		a: 1.569 m/(s <sup>2</sup> )	length: 4.124 n		
v1: 35.25 miles/h	v2: 35.46 miles/h		a: 0.354 m/(s <sup>2</sup> )	length: 4.146 n		
v1: 35.46 miles/h	v2: 35.46 miles/h		a: 0.000 m/(s <sup>2</sup> )	length: 4.149 n		
v1: 35.05 miles/h	v2: 35.46 miles/h		a: 0.706 m/(s <sup>2</sup> )	length: 4.123 n		
v1: 35.36 miles/h	v2: 35.46 miles/h		a: 0.178 m/(s <sup>2</sup> )	length: 4.133 n		
v1: 35.46 miles/h	v2: 35.25 miles/h		a: -0.357 m/(s <sup>2</sup> )	length: 4.119 n		
v1: 35.57 miles/h	v2: 35.36 miles/h		a: -0.360 m/(s <sup>2</sup> )	length: 4.114 n		
v1: 35.57 miles/h	v2: 35.25 miles/h		a: -0.541 m/(s <sup>2</sup> )	length: 4.092 n		
v1: 36.21 miles/h	v2: 35.05 miles/h		a: -2.016 m/(s <sup>2</sup> )	length: 4.099 n		

Figure 2-28. Snapshot of the screen of the remote computer.

## **3 Comparison Study with the Previous System**

### **3.1 Longer length of the Detection Area**

The length of the laser line on the road surface for detection used to be only 2.5m even if it was a combination of two laser lines, and the length of the view range on road surface of the optical system is even smaller in the previous design, i.e., less than 2m. Since the width of a highway lane normally measures 3.8m, the laser line of 2.5m and the view range of 2m make the previous design of the LBDS cannot be practically used on highway. With the new laser module and the newly designed optical system, the laser line on road surface has been enlarged to 5m, and the length of the view range of the new optical system can reach 4m. These improvements enable the LBDS to do practical detection on the highway.

### **3.2 24 channels**

In the previous design, only 8 elements of an APD array on one side are were. 8 pair of elements on both sides make 8 channels for detection. Even though the view range measures 2m, the resolution of the detection is only  $2/8=0.25\text{m/channel}$ . In the new design, the numbers of the used elements of an APD are increased to 24, making 24 channels on both sides. Considering the length of the view range has been 4m, the detection resolution of the new system becomes  $4/24=0.166\text{m/channel}$ , which is better than the previous system.

### **3.3 Stronger Anti-noise Ability**

The new design uses several anti-disturbance technologies such as hysteresis on the comparator, timing windows, and optimal self-calibration when processing the detection signal. These technologies make the LBDS obtain stronger anti-noise ability so that the LBDS can implement correct and accurate detection reliably and stably in a detection site with heavy-noise.

### **3.4 Flexibility for Different Environments**

The temperature compensation and the optimal self-calibration are firstly used in the new system design. The temperature compensation let the variation of the responsivity of APDs when the environment temperature changes to be compensated, so that the amplitude of the analog signal can be stabilized in a wide temperature range. The optimal self-calibration ensures all the DCPs are in optimal positions for correct detection. The effect of the variation on the amplitude of the analog signal, whatever it is caused by, can be eliminated by this algorithm. With these advanced technologies, the new LBDS has obtained high flexibility to apply different environments, and it has also gotten a stronger ability to operate reliably for a long term.

### **3.5 Communication**

The previous design only has two ways for transmitting the detection data to the remote computer, i.e., cable Ethernet communication and parallel communication. In the new system, wireless Ethernet communication is added. With this new communication method, the LBDS obtains mobile ability and will be more flexible to different applications.

### **3.6 Laser Alignment**

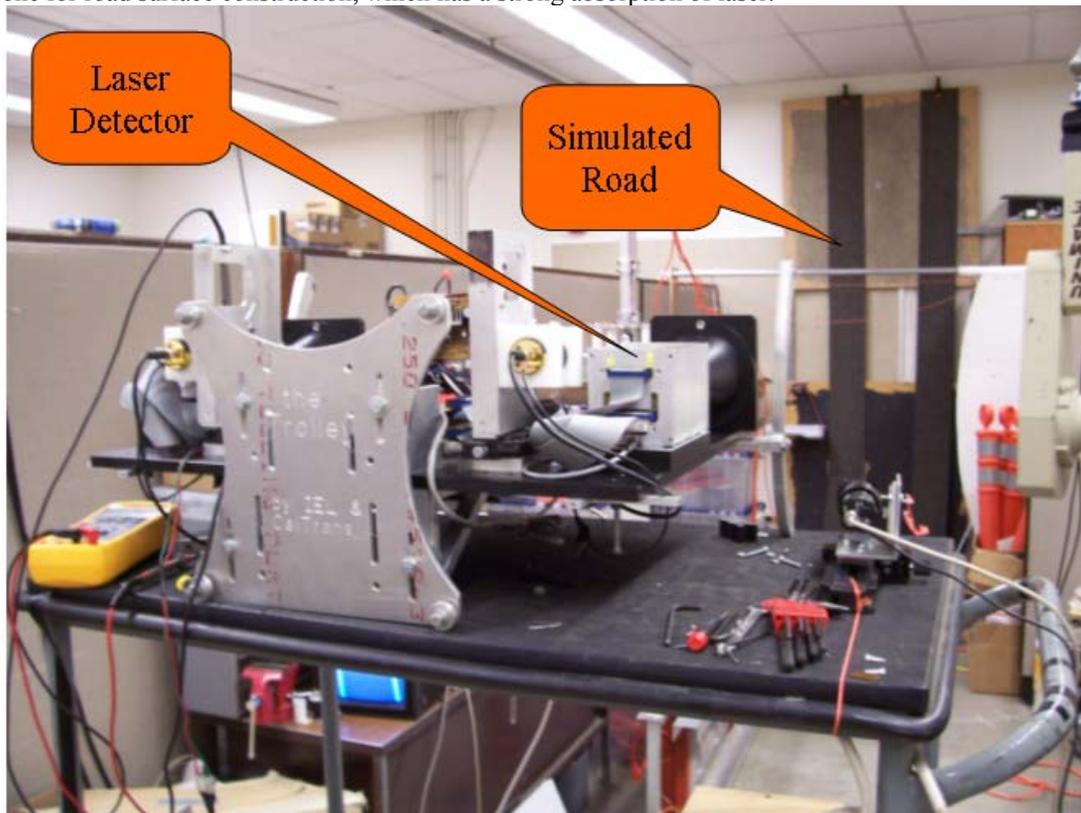
The new system has a pair of laser pointers, which are not used in the previous design, to help the alignment of the infrared laser light. This green laser pointer is very bright. It can be clearly seen even in the sunlight. With these laser pointers, the infrared laser lines can be aligned to the correct positions easily.

## 4 Testing

The system has been tested in several different ways. Indoor lab testing allows for quick and easy testing of minor changes to the LBDS. Roof testing at Bainer Hall allows for a more accurate test utilizing a vehicle traveling at low speed. A mobile truss has been created to simulate highway tests. Highway tests have been done over highway I-80 in Sacramento. The procedure to align the laser lines and adjust the electronic system is as follows.

### 4.1 Lab Testing

Testing the LBDS in the lab is important to our research. This test can be done at any time when it is necessary. We made a simulation device and environment for the test, as shown in Figure 4-1 Two black strips are put on a board next to the wall. The material covered on the strips is similar to that one for road surface construction, which has a strong absorption of laser.



**Figure 4-1. Simulation device for tests in the lab.**

The lab testing procedure is as follows:

- 1) Check whether the laser beam is in a proper position on the wall. Using the CCD camera, the laser beam can be displayed on the TV. If the laser line cannot be seen, inspect the laser module opening first. If the laser can be seen in the opening, check whether the laser line is being blocked by an obstacle. If the laser beam cannot be seen in the module opening, check the clock circuit and the 24V DC power supply.
- 2) Check whether the two laser beams are aligned in one line (both sides). Using CCD camera connect with the optical system, you can see the laser beam on the TV, then adjust the laser module to let the beam align.
- 3) Increase the aperture of the image lens to half opening.

- 4) Attach the oscilloscope probe to the output of the pre-amplifier (pin 8) and move the laser line by adjusting the screw on the rear of the laser mount until the maximum signal is obtained.
- 5) Reduce the aperture until the width of the signal is about 100ns and not saturated. .
- 6) Push the reset button to communicate with a computer.
- 7) Push the calibration button to set DCP. The output of the digital signal displayed in the computer can be checked in states of blocked or unblocked.

## **4.2 Roof Testing**

**We can set up our detector easily and conveniently on the roof of Bainer hall located on the UC Davis campus shown in Figure 4-2.**



**Figure 4-2 The roof-test site on Bainer Hall**

## **4.3 Mobile Truss Testing**

In order to avoid testing on the highway often, we can test the LBDS on a simulation site on Old Hutchinson Road in Davis. The LBDS is mounted to a mobile truss system as shown in Figure 4-3. Since it is impossible for a person to stand on the top of the truss to adjust the LBDS for different height, the LBDS is pre-adjusted to work at a distance of 5 meters in the lab and the truss is set at a height of 5 meters for testing. A remote wire button for reset and calibration is used when the LBDS is tested on the mobile truss. Communication between a computer and the LBDS is accomplished via an Ethernet cable. A single CAT-5 Ethernet cable can simply and easily extend the distance between the computer and the detector.



**Figure 4-3.** A prototype of the LBDS on a trolley and truss.



**Figure 4-4.** Field test on a roadway near UC Davis.

#### 4.4 Outdoor Highway Testing

Field tests have been conducted near the junction of highway I-5 and I-80 in Sacramento. Figure 4-5. is a picture of the test site and the detection system mounted on the highway overpass.



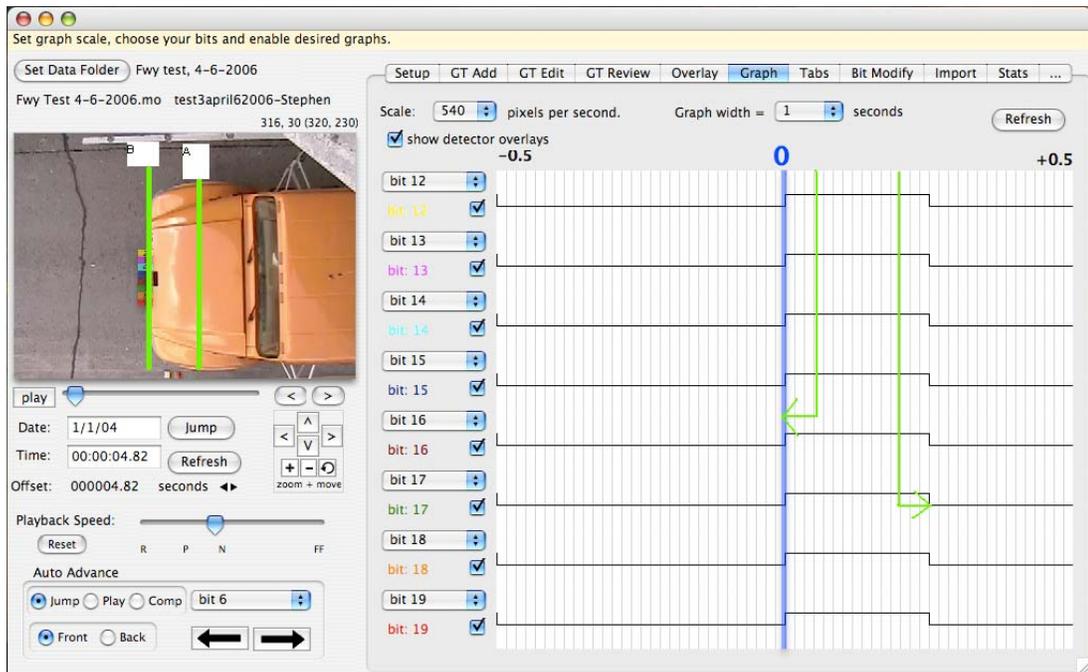
Figure 4-5. A prototype of the LBDS being tested on a highway near Sacramento.

File		Chart		Off		Reset		Rabbit		Start		UneshotOff		Continue		HLISignals		Seperate Vel																													
100 scans/10 nsec (0.1060 nsec/scan); displayed every 1 nsec																																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
v1:	35.57 miles/h	v2:	33.96 miles/h	l	a:	-2.841 m/(s <sup>2</sup> )	length:	3.929 m																																							
v1:	35.05 miles/h	v2:	34.25 miles/h	l	a:	-1.398 m/(s <sup>2</sup> )	length:	3.955 m																																							
v1:	34.15 miles/h	v2:	35.57 miles/h	l	a:	2.453 m/(s <sup>2</sup> )	length:	4.015 m																																							
v1:	34.25 miles/h	v2:	35.57 miles/h	l	a:	2.269 m/(s <sup>2</sup> )	length:	4.049 m																																							
v1:	34.35 miles/h	v2:	35.25 miles/h	l	a:	1.560 m/(s <sup>2</sup> )	length:	4.041 m																																							
v1:	34.25 miles/h	v2:	35.15 miles/h	l	a:	1.547 m/(s <sup>2</sup> )	length:	4.039 m																																							
v1:	34.25 miles/h	v2:	35.15 miles/h	l	a:	1.542 m/(s <sup>2</sup> )	length:	4.051 m																																							
v1:	34.06 miles/h	v2:	35.57 miles/h	l	a:	2.755 m/(s <sup>2</sup> )	length:	4.088 m																																							
v1:	34.45 miles/h	v2:	35.36 miles/h	l	a:	1.552 m/(s <sup>2</sup> )	length:	4.096 m																																							
v1:	34.35 miles/h	v2:	35.36 miles/h	l	a:	1.719 m/(s <sup>2</sup> )	length:	4.092 m																																							
v1:	34.74 miles/h	v2:	35.46 miles/h	l	a:	1.219 m/(s <sup>2</sup> )	length:	4.127 m																																							
v1:	34.64 miles/h	v2:	35.57 miles/h	l	a:	1.567 m/(s <sup>2</sup> )	length:	4.120 m																																							
v1:	34.64 miles/h	v2:	35.46 miles/h	l	a:	1.389 m/(s <sup>2</sup> )	length:	4.123 m																																							
v1:	34.64 miles/h	v2:	35.46 miles/h	l	a:	1.389 m/(s <sup>2</sup> )	length:	4.123 m																																							
v1:	34.95 miles/h	v2:	35.57 miles/h	l	a:	1.054 m/(s <sup>2</sup> )	length:	4.145 m																																							
v1:	34.64 miles/h	v2:	35.57 miles/h	l	a:	1.569 m/(s <sup>2</sup> )	length:	4.124 m																																							
v1:	35.25 miles/h	v2:	35.46 miles/h	l	a:	0.354 m/(s <sup>2</sup> )	length:	4.146 m																																							
v1:	35.46 miles/h	v2:	35.46 miles/h	l	a:	0.000 m/(s <sup>2</sup> )	length:	4.149 m																																							
v1:	35.05 miles/h	v2:	35.46 miles/h	l	a:	0.706 m/(s <sup>2</sup> )	length:	4.123 m																																							
v1:	35.36 miles/h	v2:	35.46 miles/h	l	a:	0.178 m/(s <sup>2</sup> )	length:	4.133 m																																							
v1:	35.46 miles/h	v2:	35.25 miles/h	l	a:	-0.357 m/(s <sup>2</sup> )	length:	4.119 m																																							
v1:	35.57 miles/h	v2:	35.36 miles/h	l	a:	0.360 m/(s <sup>2</sup> )	length:	4.114 m																																							
v1:	35.57 miles/h	v2:	35.25 miles/h	l	a:	-0.541 m/(s <sup>2</sup> )	length:	4.092 m																																							
v1:	36.21 miles/h	v2:	35.05 miles/h	l	a:	-2.016 m/(s <sup>2</sup> )	length:	4.099 m																																							

Figure 4-6. Result from a test on the highway.

The results as shown in Figure 4-6. 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 fire wire port. The user interface window contains a menu bar, a status window, and a strip chart. The menu bar is used to control 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 right to left. 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 it can display twenty-eight pairs of sensor elements.



**Figure 4-7. Snapshots of the video sync program.**

It is also noted that we have incorporated a video analysis system (Video Sync, provided by CALTrans) for analysis of the performance of the LBDS system. Figure 4-7. shows a snapshot of the Video Sync program. The Video Sync program synchronizes the video recorded during highway testing with the collected data. Since the Video Sync program takes in 60 Hz data encapsulated in the LOG\_170 format, a quasi-standard format used in data communication in CalTrans traffic management system, the 10kHz data output from the LBDS had to be reformatted. The left part of the figure shows the recorded video from highway testing. The right side shows a graph of the collected data synchronized with the video. When the video is played, the data graph will move from right to left. The green lines “A” and “B” are indicators for the laser lines projected onto the road surface. The blue 0 line shown in Figure 4-7. corresponds to the front edge of the vehicle as it passes through the Laser A beam. The blue 1 line corresponds to the front edge of the car as it passes through the Laser B beam. The blue 2 and 3 lines are the positions corresponding to the rear edge of the car as it passes through the Laser A and Laser B beams. 24 of the available 48 channels from the LBDS are visible in Figure 4-7. since the program only supports a maximum of 8 channels. The upper 4 channels in the graph are the front 0-3 channels while the bottom 4 channels are the back 0-3 channels. The Video Sync software gives a visual representation of what the system is actually capturing, giving an easier method to debug the system. It also helps in checking the real-time accuracy of the system’s data output. In the results shown, the Video Sync analysis verified that the vehicle bumpers crossed the laser sheets at about the same time.

## 5 Conclusions

We have been working on the project of creating a real-time laser-based non-intrusive detection network for the measurement of true travel time on a highway. A powerful Rabbit 3200 was introduced into our system for signal processing, synchronization between signal collection and laser pulse generation, and data communication. The power suppliers of two laser modules were separated in order to further reduce the interference between two laser modules. New circuit boards were added with plug and play ability to give the system expandability and flexibility in updating the software on the rabbit and debugging any unexpected problems. The new system main board is more compact for deploying in the field. The newly designed mechanical parts for aligning the laser beams are more efficient and easy to manipulate.

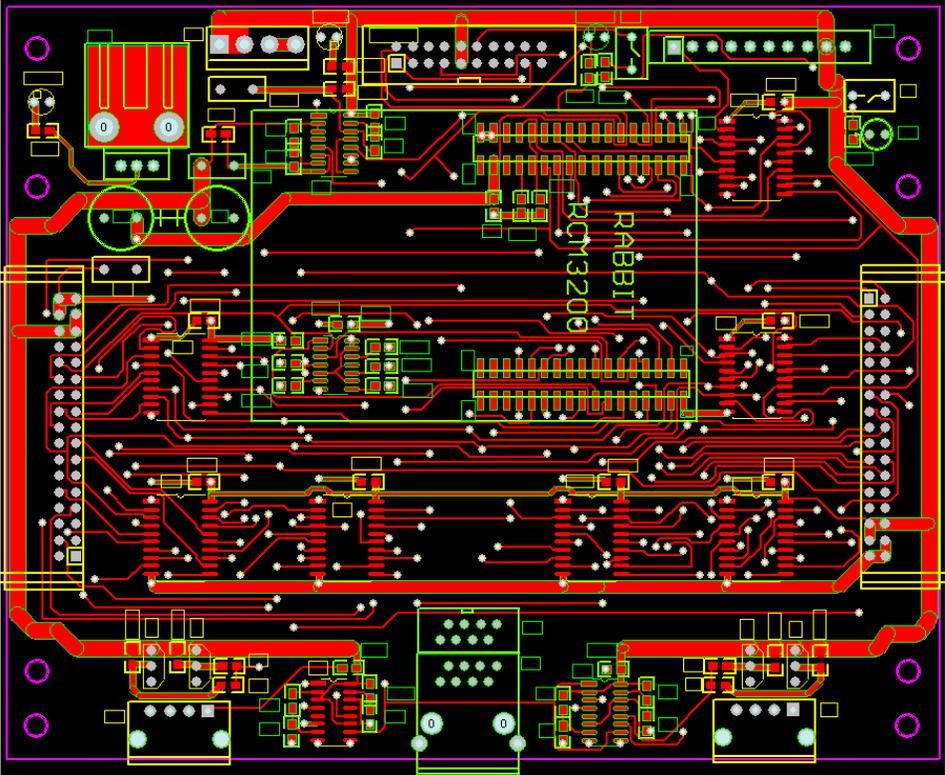
## References

- [1] J. Palen, "The Need for Surveillance in Intelligent Transportation Systems," *Intellimotion*, vol.6, no.1, pp. 1-10, Mar. 1997.
- [2] J. Palen, "The Need for Surveillance in Intelligent Transportation Systems --- Part Two," *Intellimotion*, vol.6, no.2, pp. 1-17, Jun. 1997.
- [3] "Traffic Detector – Technical Appendix," U.S. Dept. of Transportation, Federal Highway Administration, April 1985.
- [4] "Traffic Detector Handbook" U.S. Dept. of Transportation, Federal Highway Administrations, April 1985.
- [5] T. M. Hussain, T. N. Saadawi, and S. A. Ahmed, "Overhead Infrared Sensor for Monitoring Vehicular Traffic," *IEEE Trans. Vehicular Tech.*, vol. 42, pp. 477-483, Nov. 1993.
- [6] H. H. Cheng, B. D. Shaw, J. Palen, J. E. Larson, X. Hu, and K. V. Katwyk, "Non-Intrusive Laser-Based System for Detecting Objects Moving Across a Planar Surface," US Patent, No.6, 404,506, June 11, 2002.
- [7] B. Lin, H. H. Cheng, B. D. Shaw, and J. Palen, "Optical and electronic design for a field prototype of a laser-based vehicle delineation detection system," *Optics and Lasers in Engineering*, vol.36, pp.11-27, Jul. 2001.
- [8] H. H. Cheng, B. D. Shaw, J. Palen, J. E. Larson, X. Hu, and K. V. Katwyk, "A Real-Time Laser-Based Detection System for Measurement of Delineations of Moving Vehicles," *IEEE/ASME Trans. on Mechatronics*, vol.6, pp.170-187, Jun. 2001.
- [9] Z. Wang, B. Chen, H. H. Cheng, and B. D. Shaw, "Performance Analysis for Design of a High-Precision Electronic Opto-Mechanical System for Vehicle Delineation Detection on Highway," *Journal of Mechanical Design*, vol.125, pp.802-808, Dec. 2003.
- [10] H. H. Cheng, B. Shaw, J. Palen, B. Lin, B. Chen, and Z. Wang, "Development and Field Test of Laser-Based Non-Intrusive Detection System for Identification of Vehicles on the Highway," *IEEE Trans. Intelligent Transportation Systems*, Vol. 6, No. 2, pp.147-155, June 2005.
- [11] Zhaoqing Wang, Stephen S. Nestinger, Harry H. Cheng, Benjamin D. Shaw, Joe Palen. "REAL-TIME ARCHITECTURE FOR AN ELECTRO-MECH-OPTICAL SYSTEM FOR DETECTION OF VEHICLES ON HIGHWAY," *Proceedings of DETC'04:ASME 2004 International Design Engineering Technical Conferences And The Computers And Information In Engineering Conference*, Sept.28–Oct. 2, 2004 Salt Lake City, Utah USA
- [12] Zhaoqing Wang, Harry H. Cheng, Ben D. Shaw, Jeo Palen. "Noise Rejection Using Microprocessor Feedback Control for Variable S/N Ratio Pulse Signals,". *Submitted to IEEE Trans. on Instrumentation and Measurement*
- [13] Harry H. Cheng, Benjamin D. Shaw, Joe Palen, Bin Lin, Bo Chen and Zhaoqing Wang, "Development and Field Test of a Laser-Based Nonintrusive Detection System for Identification of Vehicles on the Highway," *IEEE Trans. Intelligent Transportation Systems*, 6(2), pp.147-155, Jun. 2005

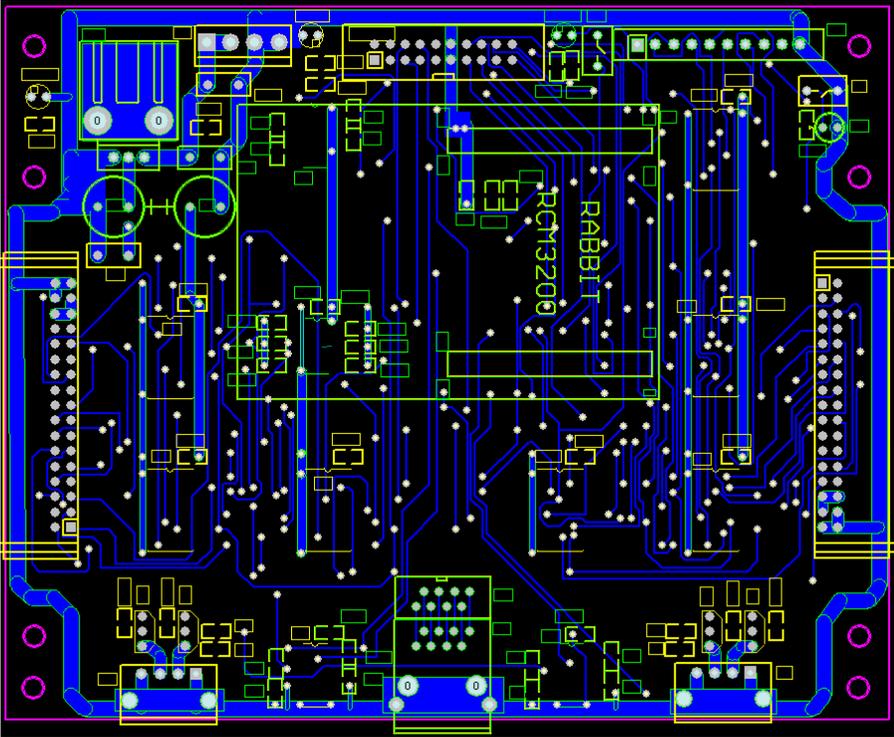
- [14] Ping Feng, Zhaoqing Wang, Harry H. Cheng, Benjamin D. Shaw, and Joe Palen, "Digitally Controlled Optimal Self-Calibration for A Laser-Photodiode Array Based Vehicle Detection System," *Proceedings of DETC/CIE'05:ASME 2005 International Design Engineering Technical Conferences and Computers and Information In Engineering Conference*, Sept.24–28, 2005 Long Beach, California, USA
- [15] A. Prokes, and V. Zeman, "Temperature Compensation of the Responsivity of Avalanche Photodiodes in Free-Space Optical Communication Systems," *The IEEE-Siberian Conference on Control and Communications*, 2003. SIBCON 2003.
- [16] Halvorson, G. A., "Automated Real-Time Dimension Measurement of Moving Vehicles Using Infrared Laser Rangefinders", MS Thesis, University of Victoria, 1995.
- [17] 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.
- [18] U. Moon and B. Song, "Background Digital Calibration Techniques for Pipelined ADC's," *IEEE Trans. Circuits Syst. II*, vol. 44, pp. 102-109, Feb. 1997.
- [19] V. Liberali, F. Cherchi, L. Disingrini, M. Gottardi, S. Gregori, and G. Torelli, "A Digital Self-Calibration Circuit for Absolute Optical Rotary Encoder Microsystems," *IEEE Trans. Instrum. Meas.*, vol. IM-52, pp. 149-157, Feb. 2003.
- [20] J. Guo, W. Law, W. J. Helms, and D. J. Allstot, "Digital Calibration for Monotonic Pipelined A/D Converters," *IEEE Trans. Instrum. Meas.*, vol. IM-53, pp. 1485-1492, Dec. 2004.

# Appendix A: Circuit and Board Diagrams

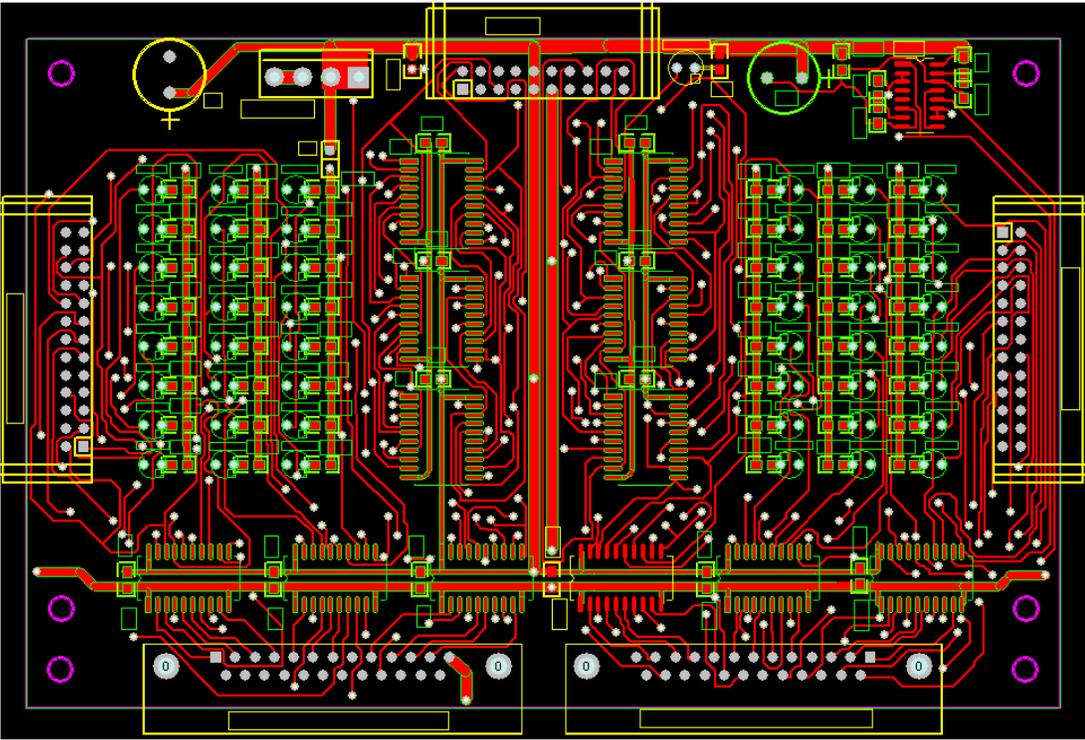
Main control board top layer



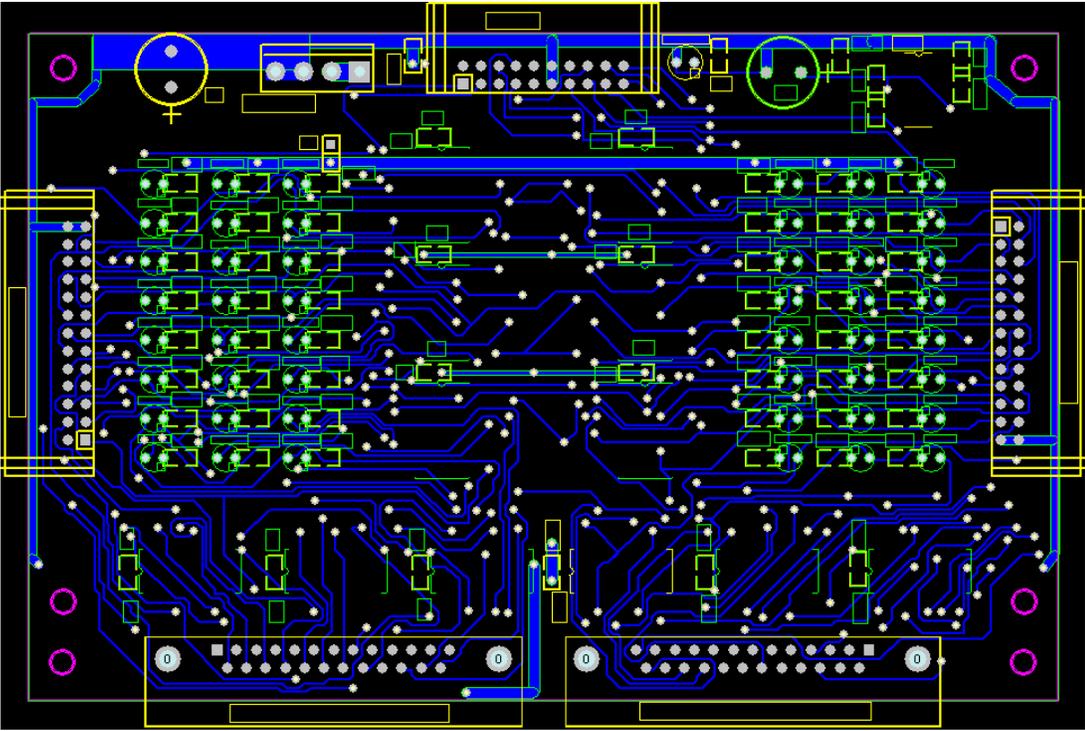
Main control board bottom layer



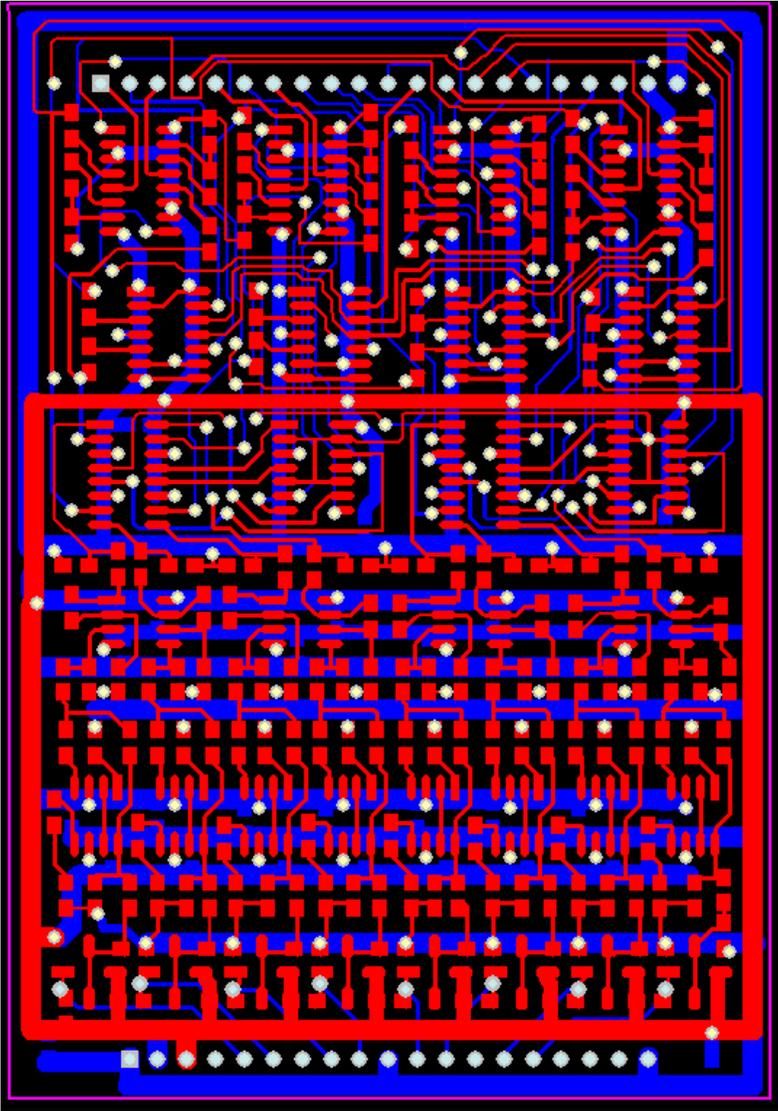
Main signal board top layer



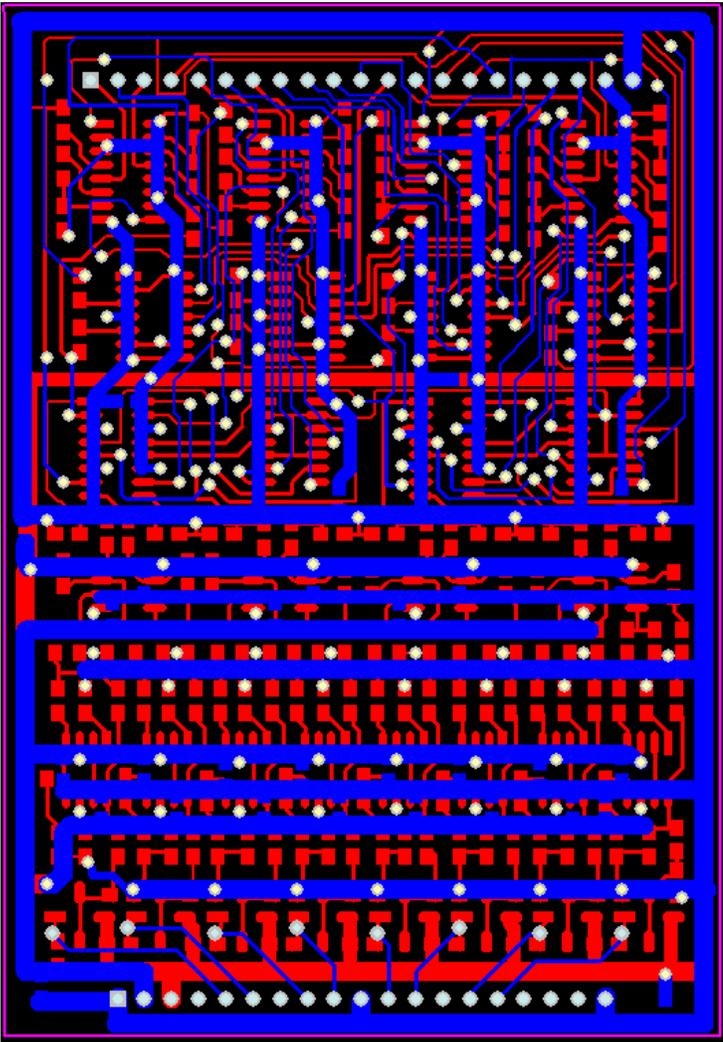
Main signal board bottom layer



Signal preamplifier and shaping board top layer



Signal preamplifier and shaping board bottom layer



## Appendix B: Source Code

### Source Code in rabbit 3200 for control and communication

```
#class auto
/*****
 * Pick the predefined TCP/IP configuration for this sample. See
 * LIB\TCPIP\TCP_CONFIG.LIB for instructions on how to set the
 * configuration.
 * Config #100 = lbsd_config.lib
 *****/

/* Defines the size of the buffer that is used to send TCP datagrams */
#define ETH_MTU 1000
#define BUFF_SIZE (ETH_MTU-40)*6 //must be smaller than (ETH_MTU - (IP Datagram + TCP Segment))
#define TCP_BUF_SIZE ((ETH_MTU-40)*10) // sets up (ETH_MTU-40)*6 bytes for Tx & Rx buffers
#define TCPCONFIG 100
#define TCP_OPENTIMEOUT 100
//define RETRAN_STRAT_TIME 12

/* Defines the interval(in seconds) to send out TCP packet */
#define HEARTBEAT_RATE 5

/* Defines the local TCP port to send packets */
#define LOCAL_PORT 2801

/* Where to send the heartbeats. If it is set to all '255's, it will be a broadcast packet */
#define REMOTE_IP "10.0.0.11"

/* the destination port to send to */
#define REMOTE_PORT 2002

#memmap xmem

#define D_SIZE 100
#define D_CHANNELS 6
#define PWM_FREQUENCY 10000

/* define the same as the control.h in xvehicle */

#define START_TASK 1
#define STOP_TASK 2
#define PLAY_TASK 3
#define RESET_TASK 4
#define CALIBRATE_TASK 5
#define CLOSE_TASK 6
#define TROLLEY_STEPLEFT 7
#define TROLLEY_STEPRIGHT 8
#define TROLLEY_MOVELEFT 9
#define TROLLEY_MOVERIGHT 10
#define TROLLEY_STOP 11
#define TROLLEY_LOCK 12
#define TROLLEY_UNLOCK 13
#define TROLLEY_ACCELERATE 14
#define TROLLEY_DEACCELERATE 15

#include "dertcp.lib"
/* define storage for the sampling data */
char data0[D_SIZE*D_CHANNELS+10];
char data1[D_SIZE*D_CHANNELS+10];
char *data_array_ptr;
```

```

/* Define storage for calibration */
char XDCCP_max[50], XDCCP_min[50];           // 0-23 for left side, 24-47 for right side
char testdata_left[3], testdata_right[3];   // storage for read data when calibration
char *testdata_left_ptr, *testdata_right_ptr; // pointers of the variables
tcp_Socket tsock;
int count;
char flg, intflag;           // intflag to indicate Rabbit has been interrupted
char tcpDataSendFlg;        // tcpDataSendFlg to indicate the data stored in the memory is ready to send
char U12Shadow;
char send_num;
int sequence1;

/*****
 * Function Declaration
 *
 *****/
void readData();
void outData();
int sendPacket(char *buf);

void setPortInit();
void setDataInit();
void setInterrupt();
void setPwm(int requery);
void setHDCP();
void printResults();
void dispData(char *buf);
int openTcpSock();

void setXDCCP();
void calibration();
void errorIndicate();

/*****
 * main() function
 *
 *****/
int main() {
    int cal_key;
    char oldflg, temp;
    unsigned char command[1];
    setPortInit();
    setDataInit();
    setPwm(PWM_FREQUENCY);
    setInterrupt();
    testdata_left_ptr = testdata_left; //save the address in the pointer
    testdata_right_ptr = testdata_right; //save the address in the pointer

    setXDCCP();

    tcpDataSendFlg = 0;
    oldflg=0;
    setHDCP();
    if(ifstatus(IF_ETH0))
        printf("ETH0 is up now.\n");

    while ( 1 ) {
        tcp_tick(&tsock);
        //open a TCP socket for communication
        if(!sock_established(&tsock)){
            if(openTcpSock(>0)

```

```

    sock_mode(&tsock, TCP_MODE_ASCII);
}

if(BitRdPortI(PCDR, 1)==0) {
    costate {
        waitfor( DelayMs( 10L ) );
        if(BitRdPortI(PCDR, 1)==0) {
            while(1) {
                if(BitRdPortI(PCDR, 1)==1) {
                    calibration();
                    break;
                }
            }
        }
    }
}

if(sock_established(&tsock)&& sock_bytesready(&tsock)){
    if(sock_ared(&tsock, command, 1)){
        switch (command[0]){
            case CALIBRATE_TASK:
                printf("calibration\n");
                U12Shadow &= 0xe0;
                WrPortI(PGDR, &PGDRShadow, U12Shadow); // set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in
                                                         // U12Shadow

                BitWrPortI(PCDR, &PCDRShadow, 1, 2);
                BitWrPortI(PCDR, &PCDRShadow, 0, 2); // stop trolley at first
                calibration();
                break;
            case RESET_TASK:
                printf("reset\n");
                forceSoftReset();
                break;
            case TROLLEY_STEPLEFT:
                printf("trolley step moving left\n");
                U12Shadow &= 0xe0;
                WrPortI(PGDR, &PGDRShadow, U12Shadow); // set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in
                                                         //U12Shadow

                BitWrPortI(PCDR, &PCDRShadow, 1, 2);
                BitWrPortI(PCDR, &PCDRShadow, 0, 2); // stop trolley at first
                U12Shadow |= 0x01;
                WrPortI(PGDR, &PGDRShadow, U12Shadow); // set PG4=0, PG3=0, PG2=0, PG1=0, PG0=1
                BitWrPortI(PCDR, &PCDRShadow, 1, 2);
                BitWrPortI(PCDR, &PCDRShadow, 0, 2); // set Q1=1 for trolley step move left
                break;
            case TROLLEY_STEPRIGHT:
                printf("trolley step moving right\n");
                U12Shadow &= 0xe0;
                WrPortI(PGDR, &PGDRShadow, U12Shadow); // set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in
                                                         //U12Shadow

                BitWrPortI(PCDR, &PCDRShadow, 1, 2);
                BitWrPortI(PCDR, &PCDRShadow, 0, 2); // stop trolley at first
                U12Shadow |= 0x04;
                WrPortI(PGDR, &PGDRShadow, U12Shadow); // set PG4=0, PG3=0, PG2=1, PG1=0, PG0=0
                BitWrPortI(PCDR, &PCDRShadow, 1, 2);
                BitWrPortI(PCDR, &PCDRShadow, 0, 2); // set Q3=1 for trolley step move right
                break;
            case TROLLEY_MOVELEFT:
                printf("trolley moving left\n");
                U12Shadow &= 0xe0;
                WrPortI(PGDR, &PGDRShadow, U12Shadow); // set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in
                                                         // U12Shadow
        }
    }
}

```

```

BitWrPortI(PCDR, &PCDRShadow, 1, 2);
BitWrPortI(PCDR, &PCDRShadow, 0, 2);
U12Shadow |= 0x11;
WrPortI(PGDR, &PGDRShadow, U12Shadow);
BitWrPortI(PCDR, &PCDRShadow, 1, 2);
BitWrPortI(PCDR, &PCDRShadow, 0, 2);
break;
case TROLLEY_MOVERIGHT:
    printf("trolley moving right\n");
    U12Shadow &= 0xe0;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);

    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    U12Shadow |= 0x14;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);
    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    break;
case TROLLEY_LOCK:
    printf("trolley brake\n");
    U12Shadow &= 0xe0;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);

    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    U12Shadow |= 0x08;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);
    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    break;
case TROLLEY_UNLOCK:
    printf("trolley unbrake\n");
    U12Shadow &= 0xe0;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);

    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    U12Shadow |= 0x02;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);
    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    break;
case TROLLEY_ACCELERATE:
    printf("trolley speed up\n");
    U12Shadow &= 0xe0;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);

    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    U12Shadow |= 0x18;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);
    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);
    break;
case TROLLEY_DEACCELERATE:
    printf("trolley speed down\n");
    U12Shadow &= 0xe0;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);

    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);

```

// stop trolley at first

// set PG4=1, PG3=0, PG2=0, PG1=0, PG0=1

// set Q5=1 & Q1=1 for trolley continous move left

// set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in //U12Shadow

// stop trolley at first

// set PG4=1, PG3=0, PG2=1, PG1=0, PG0=0

// set Q5=1 & Q3=1 for trolley continous move right

// set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in // U12Shadow

// stop trolley at first

// set PG4=0, PG3=1, PG2=0, PG1=0, PG0=0

// set Q4=1 for trolley locking, brake goes up

// set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in //U12Shadow

// stop trolley at first

// set PG4=0, PG3=0, PG2=0, PG1=1, PG0=0

// set Q2=1 for trolley unlocking, brake goes down

// set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in // U12Shadow

// stop trolley at first

// set PG4=1, PG3=1, PG2=0, PG1=0, PG0=0

// set Q5=1 & Q4=1 for trolley speed accelerated

// set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in //U12Shadow

// stop trolley at first

```

    U12Shadow |= 0x12;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);           // set PG4=1, PG3=0, PG2=0, PG1=1, PG0=0
    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);             // set Q5=1 & Q2=1 for trolley speed deaccelerated
    break;
case TROLLEY_STOP:
    printf("trolley stop!!\n");
    U12Shadow &= 0xe0;
    WrPortI(PGDR, &PGDRShadow, U12Shadow);           // set PG4-PG0=0, let Q5-Q1 of U12 =0, store the output in
                                                         //U12Shadow
    BitWrPortI(PCDR, &PCDRShadow, 1, 2);
    BitWrPortI(PCDR, &PCDRShadow, 0, 2);             // stop trolley
    break;
default:
    break;
}
}
}
printf("testdata_left= %x %x %x\t\t", testdata_left[0], testdata_left[1], testdata_left[2]);
printf("testdata_right= %x %x %x\n", testdata_right[0], testdata_right[1], testdata_right[2]);

if (tcpDataSendFlg) {
    tcpDataSendFlg = 0;
    //printf("seq:%d\n",sequence1);
    switch (flg) {
        case 0:                //data1 is ready to be sent
            if(sock_established(&tsock))
                sendPacket(data1);
            break;
        case 1:                //data0 is ready to be sent
            if(sock_established(&tsock))
                sendPacket(data0);
            break;
        case 2:
            //adjust potentiometer
            //setXDPC();
            break;
        default:
            break;
    }
    // end of switch( flg )
}
// end of if(tcpDataSendFlg==1)
}
// end of while()
}

/*****
* setHDCP() For DHCP setup
*
* A very basic TCP/IP program, that will initilize the TCP/IP interface,
* if set TCPCONFIG as 3, DHCP (Dynamic Host Configuration Protocol) or
* BOOTP (Bootstrap Protocol) will be used to obtain IP addresses and other
* network configuration items.
* Otherwise, using the static IP address for this rabbit.
*****/
void setHDCP() {
    int status;
    #if TCPCONFIG==3
        // Set runtime control for sock_init()...
        ifconfig(IF_DEFAULT, // (DHCP only works on default interface)
            IFS_DHCP_TIMEOUT, 6, // Specify timeout in seconds
            IFS_DHCP_FALLBACK, 1, // Allow use of fallbacks to static configuration
            IFS_ICMP_CONFIG, 1, // Also allow use of directed ping to configure (only if DHCP times out).
            IFS_UP,

```

```

IFS_END);
printf("Starting network (max wait %d seconds)...\\n", _bootptimeout);
#endif

status = sock_init();

switch (status) {
case 1:
    printf("Could not initialize packet driver.\\n");
    exit(1);
case 2:
    printf("Could not configure using DHCP.\\n");
    break; // continue with fallbacks
case 3:
    printf("Could not configure using DHCP; fallbacks disallowed.\\n");
    exit(3);
default:
    break;
}
#endif TCPCONFIG==3
if (_dhcp host != ~0UL)
    printf("Lease obtained\\n");
else {
    printf("Lease not obtained. DHCP server may be down.\\n");
    printf("Using fallback parameters...\\n");
}
#endif
printResults();
}

void outData(char flg) {
    BitWrPortI(PCDR, &PCDRShadow, flg, 4);
}

/*****
 * dispData(char *buf)
 * the buf is a data read by readData in interrupt
 * this function typically used as debug
 *****/
void dispData(char *buf){
    static int i;
    printf("flg=%d\\n", flg);
    for( i = 1; i <= D_CHANNELS*D_SIZE; i++){
        printf(" %d", (unsigned)buf[i-1]);
        if( i%D_CHANNELS == 0 )
            printf("\\n");
    }
}

/*****
 * setPortInit() set ports used as input or output
 *****/
void setPortInit() {
    WrPortI(SPCR, &SPCRShadow, 0x80); // set port A as all inputs, p126

    WrPortI(PBDDR, &PBDDRShadow, 0xff); // set port B as all outputs, p127
    WrPortI(PBDR, &PBDRShadow, 0xff); // set PB7-PB0=1

    WrPortI(PCDR, &PCDRShadow, 0xea); // set PC0=0, PC2=0, PC4=0, others="1", for TXD use 7407
}

```

```

BitWrPortI(PDDDR, &PDDDRShadow, 1, 4); // set PD4 as output
BitWrPortI(PDDDR, &PDDDRShadow, 1, 5); // set PD5 as output
WrPortI(PDDCR, &PDDCRShadow, 0x00); // set port D as standard output, not open drain
//BitWrPortI(PDDCR, &PDDCRShadow, 0, 4); // set PD4 as standard output, not open drain
//BitWrPortI(PDDCR, &PDDCRShadow, 0, 5); // set PD5 as standard output, not open drain
BitWrPortI(PDDR, &PDDRShadow, 1, 4); // set PD4=1, enable U3B work
BitWrPortI(PDDR, &PDDRShadow, 1, 5); // set PD5=1, enable U3A work

WrPortI(PEDR, &PEDRShadow, 0x00); // set port E outputs PE7-PE1 = 0
WrPortI(PEDDR, &PEDDRShadow, 0xfe); // set PE7-PE1 as outputs, PE0 as inputs for the interrupt
WrPortI(PECR, &PECRShadow, 0x00); // set PECR all bits as 0, for all outputs are clocked

WrPortI(PFDDR, &PFDDRShadow, 0xff); // set port F as all outputs
WrPortI(PFDCR, &PFDCRShadow, 0x00); // set port F as all standard output, not open drain
WrPortI(PFDR, &PFDRShadow, 0x01); // set PF7-PF1=0, PF0=1, but PF7 is used as PWM
WrPortI(PFFR, &PFFRShadow, 0x80); // set PF7=PWM

WrPortI(PGDDR, &PGDDRShadow, 0xff); // set port G as all outputs
WrPortI(PGDR, &PGDRShadow, 0xff); // set all output of port G as "1", PG0-PG7=1

BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC(CLKD) of XDCP to 1
BitWrPortI(PCDR, &PCDRShadow, 1, 0); // set PC0=1, U/D(TXD) of XDCP = "1", count up

WrPortI(PGDR, &PGDRShadow, 0xff); // set all output of port G as "1", PG0-PG7=1
WrPortI(PFDR, &PFDRShadow, RdPortI(PFDR) | 0x7e); // set PF6-PF1=1, select all 6 hct373
WrPortI(PFDR, &PFDRShadow, RdPortI(PFDR) & 0x81); // set PF6-PF1=0, set all 48 CS of DCPs to 1

WrPortI(PGDR, &PGDRShadow, 0x00); // set all output of port G as "0", PG0-PG7=0
BitWrPortI(PCDR, &PCDRShadow, 1, 2); // set PC2=1
BitWrPortI(PCDR, &PCDRShadow, 0, 2); // set PC2=0, set all outputs of U12 "0", (trolley control)
U12Shadow = 0; // initialize the shadow of U12
}

/*****
* initializing the interrupter
* set the external INT0
* and interrupter handler as readData()
*****/
void setInterrupt() {
//enable external INT0 on PE0, set the interrupt routin to readData
SetVectExtern3000(0, readData);
//refer to p_95 00001001
WrPortI(I0CR, &I0CRShadow, 0x05); // enable external INT0 on PE0,
// falling edge, priority 1
WrPortI(I1CR, &I1CRShadow, 0x00); // disable external INT1 on PE1,
//
data_array_ptr = data0; // initialize the pointer pointing to data0 array
flg = 0; // initialize the flg
tcpDataSendFlg = 0; // initialize the flag for indicating sending the data
}

/*****
* initializing PWM generator
* set the PWM frequency and the duty cycle
* define the output pin
*****/
void setPwm(int frequency){
unsigned long freq;
int pwm_options;

// request frequency PWM cycle (will select closest possible value)

```

```

freq = pwm_init(frequency);
printf("Actual PWM frequency = %lu Hz\n", freq);

pwm_options = 0;

pwm_set(3, 0.1 * 1024, pwm_options); //set PF7 as the PWM output port with 10% duration
}

/*****
*   initializing Data storage area
*   define the memory area to store the data
*****/
void setDataInit() {
    int i;
    /*
    for( i=0; i<D_SIZE*D_CHANNELS; i+=3){
        data0[i]=0xff;
        data0[i+1]=0xff;
        data0[i+2]=0xff;
        data1[i]=0x00;
        data1[i+1]=0x00;
        data1[i+2]=0x00;
    }
    */
    memset(data0, 0x55, D_SIZE*D_CHANNELS+1);
    memset(data1, 0xff, D_SIZE*D_CHANNELS+1);

    send_num =0x31;
    count = 0;
    flg = 0;
    sequence1 = 0;
}

/***** interrupt routines *****/
*
*   readData(),
*
*   use PE0 as INT. the data will be read from the input port
*
*   Data read from PORT A, restore in the data0[D_CHANNELS*D_SIZE] when flg=0
*           data1[D_CHANNELS*D_SIZE] when flg=1
*   Each interrupt reads the PORT A for D_CHANNELS times to get 48 channels from
*   input data buffer. the PORT B is used to select which 8 channels
*   will be read:
*
*   for 0-7 channels, set PB2 as 0 (left 1-8)
*   8-15 channels, set PB3 as 0 (left 9-16)
*   16-23 channels, set PB4 as 0 (left 17-24)
*
*   24-31 channels, set PB5 as 0 (right 1-8)
*   32-39 channels, set PB6 as 0 (right 9-16)
*   40-47 channels, set PB7 as 0 (right 17-24)
*
*****/
nodebug root interrupt void readData() {
    intflag=1;

    //read data from PORT A and store them into data array ( data0[] or data1[] )
    //outData(0);

    #asm

```

```

;select left side 1-8 channels
ioi ld a, (PBDR)
or 0xfc ; set PB7-PB2 = "1", not select all 6 hc244
and 0xfb ; set PB2 to 0, select hct244 for left side channel 1-8
ioi ld (PBDR), a

ld hl, (data_array_ptr) ; get the address of the data array working at moment

; dataa = A7 A6 A5 A4 A3 A2 A1 A0
; channel: 8 7 6 5 4 3 2 1
ioi ld a, (PADR) ; get the data from the port A
ld (hl), a ; store the data to a unit in data array

ld de, (testdata_left_ptr) ; get the address of testdata_left[0]
ld (de), a ; save the data to testdata_left[0]

;select left side 9-16 channels
ioi ld a, (PBDR)
or 0xfc ; set PB7-PB2 = "1", not select all 6 hc244
and 0xf7 ; set PB3 to 0, select hct244 for left side channel 9-16
ioi ld (PBDR), a

inc hl ; get the address of next unit in data array

; dataa = A7 A6 A5 A4 A3 A2 A1 A0
; channel: 16 15 14 13 12 11 10 9
ioi ld a, (PADR) ; get the data from the port A
ld (hl), a ; store the data to the next unit in data0[]

;ld de, (testdata_left_ptr+1) ; get the address of testdata_left[1]
inc de
ld (de), a ; save the data to testdata_left[1]

;select left side 17-24 channels
ioi ld a, (PBDR)
or 0xfc ; set PB7-PB2 = "1", not select all 6 hc244
and 0xef ; set PB4 to 0, select hct244 for left side channel 17-24
ioi ld (PBDR), a

inc hl ; get the address of next unit in data array

; dataa = A7 A6 A5 A4 A3 A2 A1 A0
; channel: 24 23 22 21 20 19 18 17
ioi ld a, (PADR) ; get the data from the port A
ld (hl), a ; store the data to the next unit in data0[]

;ld de, (testdata_left_ptr+2) ; get the address of testdata_left[2]
inc de
ld (de), a ; save the data to testdata_left[2]

;select right side 1-8 channels
ioi ld a, (PBDR)
or 0xfc ; set PB7-PB2 = "1", not select all 6 hc244
and 0xdf ; set PB5 to 0, select hct244 for right side channel 1-8
ioi ld (PBDR), a

inc hl ; get the address of next unit data array

; dataa = A7 A6 A5 A4 A3 A2 A1 A0
; channel: 8 7 6 5 4 3 2 1

```

```

ioi ld a, (PADR)      ; get the data from the port A
ld (hl), a           ; store the data to the next unit in data0[]

ld de, (testdata_right_ptr) ; get the address of testdata_right[0]
ld (de), a           ; save the data to testdata_right[0]

;select right side 9-16 channels
ioi ld a, (PBDR)
or 0xfc              ; set PB7-PB2 = "1", not select all 6 hc244
and 0xbf             ; set PB6 to 0, select hct244 for right side channel 9-16
ioi ld (PBDR), a

inc hl               ; get the address of next unit in data array

; dataa = A7 A6 A5 A4 A3 A2 A1 A0
; channel: 16 15 14 13 12 11 10 9
ioi ld a, (PADR)    ; get the data from the port A
ld (hl), a          ; store the data to the next unit in data0[]

;ld de, (testdata_right_ptr+1) ; get the address of testdata_right[1]
inc de
ld (de), a          ; save the data to testdata_right[1]

;select right side 17-24 channels
ioi ld a, (PBDR)
or 0xfc              ; set PB7-PB2 = "1", not select all 6 hc244
and 0x7f             ; set PB7 to 0, select hct244 for right side channel 17-24
ioi ld (PBDR), a

inc hl               ; get the address of next unit in data array

; dataa = A7 A6 A5 A4 A3 A2 A1 A0
; channel: 24 23 22 21 20 19 18 17
ioi ld a, (PADR)    ; get the data from the port A
ld (hl), a          ; store the data to the next unit in data0[]

;ld de, (testdata_right_ptr+2) ; get the address of testdata_right[2]
inc de
ld (de), a          ; save the data to testdata_right[2]

inc hl
ld (data_array_ptr),hl ; store the position pointer

ioi ld a, (PBDR)
or 0xfc              ; set PB7-PB2 = "1", not select all 6 hc244
ioi ld (PBDR), a    ;
#endasm

count += D_CHANNELS;
if(count >= D_CHANNELS*D_SIZE){
sequence1++;
count = 0;
tcpDataSendFlg = 1; // set flag to indicate the data is ready to be sent
if(flq){
flg = 0;
data_array_ptr = data0;
}
else{
flg = 1;
data_array_ptr = data1;
}
}
}

```

```

}

/*****
 * Print some of the DHCP parameters received.
 *****/
static void printResults(void) {
    auto long tz;
    auto word i;

    printf("Network Parameters:\n");
    printf(" My IP Address = %08lX\n", my_ip_addr);
    printf(" Netmask = %08lX\n", sin_mask);
    #if TCPCONFIG==3
    if (_dhcphost != ~0UL) {
        if (_dhcpstate == DHCP_ST_PERMANENT) {
            printf(" Permanent lease\n");
        } else {
            printf(" Remaining lease = %ld (sec)\n", _dhcplife - SEC_TIMER);
            printf(" Renew lease in %ld (sec)\n", _dhcpt1 - SEC_TIMER);
        }
        printf(" DHCP server = %08lX\n", _dhcphost);
    }
    #endif
    if (gethostname(NULL,0))
        printf(" Host name = %s\n", gethostname(NULL,0));
    if (getdomainname(NULL,0))
        printf(" Domain name = %s\n", getdomainname(NULL,0));
    #if TCPCONFIG==3
    if (dhcp_get_timezone(&tz))
        printf(" Timezone (fallback only) = %ldh\n", tz / 3600);
    else
        printf(" Timezone (DHCP server) = %ldh\n", tz / 3600);
    #endif
    for (i = 0; i < * _last_nameserver; i++)
        printf(" DNS server #%u = %08lX\n", i+1, def_nameservers[i]);
    ip_print_ifs();
    router_printall();
}

/*****
 * int sendPacket(char *buf)
 *
 * Send data to the remote Data. the size of package is 601.
 * It should consider the situation:
 * that the data can't send out because the send-buffer is full.
 *
 *****/
int sendPacket(char *buf) {
    static long sequence;
    auto int length, retval;
    auto int lten;
    static char buf_last[D_SIZE*D_CHANNELS];
    char zero_buf[D_SIZE*D_CHANNELS];
    #GLOBAL_INIT
    {
        sequence = 0;
    }
    memset(zero_buf, 0xFF, D_SIZE*D_CHANNELS);
    sequence++;
    length = D_SIZE*D_CHANNELS;

```

```

/*
if(strncmp(buf, zero_buf,length)==0)
  if( strncmp(buf, buf_last,length)==0)
    return 1;
*/
sock_flushnext(&tsock);
tcp_tick(&tsock);
/* there is enough room in the buffer
   * before adding data with a blocking function.*/
if( sock_tbleft(&tsock) < length) {
  sock_abort(&tsock);
  return 0;
}
retval = sock_awrite(&tsock, buf, length);
//printf("sendout:%d\n",retval);
switch (retval) {
case 0:
  printf("Insufficient free space in transmit buffer!\n");
  break;
case 1:
  printf("Len is greater than the total socket receive buffer size!\n");
  break;
case 2:
  printf("The socket has been closed for further transmission!\n");
  break;
case 3:
  printf("len<0 or parameter wa invalid!\n");
  break;
}
sock_flush(&tsock);
//strncpy(buf_last,buf,length);
return 1;
}

/*****
* int openTcpSock()
*
* open a tcp socket
* return 1 successfully
* return -1 unable to open TCP session
*****/
int openTcpSock(){

  //printf("seq:%d\n",sequence1);
  sock_abort(&tsock);
  //printf("try to open socket ... \n");
  if( !tcp_open( &tsock, 0, resolve(REMOTE_IP), REMOTE_PORT , NULL )) {
    puts("Unable to open TCP session!\n");
    return -1;
  }
  else if (sock_established(&tsock)) {
    printf("Established OK!\n");
    return 1;
  }
  else {
    //printf(" Failed to establish a socket!\n");
    return -1;
  }
}

/*****
* void setXDSCP()

```

```

*
* initialize all the XDCP
*****/
void setXDCP() {
    int i;
    char value;

    WrPortI(PGDR, &PGDRShadow, 0x00);    //set PG7-PG0 = 00000000 ( select odd 24 DCPs, 1,3,5...47 )
    value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, value);    //set PF6-PF1=1, select all 6 hct373
    value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, value);    //set PF6-PF1=0, set odd 24 CS of DCPs of total 48 DCPs to 0
                                        // ( using AD5222 )

    BitWrPortI(PCDR, &PCDRShadow, 1, 0);    //set PC0=1, U/D(TXD) of XDCP to count up
    for(i=0;i<130;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);    //set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);    //set INC(CLKD) of XDCP to 1
    }
    //set XDCP to top
    BitWrPortI(PCDR, &PCDRShadow, 0, 0);    //set PC0=0, U/D(CLKD) of XDCP to count down
    for(i=0;i<77;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);    //set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);    //set INC(CLKD) of XDCP to 1
    }
    //set XDCP to midway, Vw=2.0V

    WrPortI(PGDR, &PGDRShadow, 0xaa);    //set PG7-PG0 = 10101010 ( select even 24 DCPs, 2,4,6...48 )
    value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, value);    //set PF6-PF1=1, select all 6 hct373
    value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, value);    //set PF6-PF1=0, set even 24 CS of DCPs of total 48 DCPs to 0
                                        // ( using AD5222 )

    BitWrPortI(PCDR, &PCDRShadow, 1, 0);    //set PC0=1, U/D(TXD) of XDCP to count up
    for(i=0;i<130;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);    //set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);    //set INC(CLKD) of XDCP to 1
    }
    //set XDCP to top
    BitWrPortI(PCDR, &PCDRShadow, 0, 0);    //set PC0=0, U/D(CLKD) of XDCP to count down
    for(i=0;i<77;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);    //set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);    //set INC(CLKD) of XDCP to 1
    }
    //set XDCP to midway, Vw=2.0V

    BitWrPortI(PCDR, &PCDRShadow, 1, 0);    //set PC0=1, U/D(TXD) of XDCP to count up
    WrPortI(PGDR, &PGDRShadow, 0xff);    //set PG7-PG0 = 11111111
    value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, value);    //set PF6-PF1=1, select all 6 hct373
    value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, value);    //set PF6-PF1=0, set all 24 CS of 24 DCP chips to 1
}

/*****
* void calibration()
*
* calibrate all the XDCP to optimal position
*
*****/
void calibration() {
    char Step_counter;    // step counter for XDCP
    char XDCP_max[48], XDCP_min[48], XDCP_set[48];    // 0-23 for left side, 24-47 for right side, (_max & _min
                                                // for maximum and minimum step count, _set for setting of XDCP )
    char XDCPSetDone_max[48], XDCPSetDone_min[48];    // store the symbol of the channel which has been set

```

```

char samplingData[12][6];           // store the sample data of 10-12 times, [[0]-[2] for left side, [[3]-[5] for
    right side

//char leftXDCP_max[8], leftXDCP_min[8];           // store maximum and minimum step count for left XDCP
//char rightXDCP_max[8], rightXDCP_min[8];         // store maximum and minimum step count for right XDCP
//char leftXDCPset[8], rightXDCPset[8];           // save the setting of XDCP
char temp1, temp2, temp3;
int i, j, symbol;
char value;

    memset(samplingData, 0x00, 12*6);

    BitWrPortI(PBDR, &PBDRShadow, 0, 0);           // set PB0=0, turn on the LED indicator

    WrPortI(PGDR, &PGDRShadow, 0x00);             // set PG7-PG0 = 00000000 (select odd 24 DCPs, using AD5222)
    value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, value);           // set PF6-PF1=1, select all 6 hct373
    value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, value);           // set PF6-PF1=0, set 24 odd CS of all 48 DCPs to 0

    BitWrPortI(PFDR, &PFDRShadow, 1, 0);         // set INC(CLKD) of XDCP to 1
    BitWrPortI(PCDR, &PCDRShadow, 1, 0);         // set PC0=1, U/D(TXD) of XDCP to "1", count up
    for(i=0;i<130;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);     // set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);     // set INC of XDCP to "1", generate a negative pulse on INC
line
    }
        // set XDCP to top
    BitWrPortI(PCDR, &PCDRShadow, 0, 0);         // set PC0=0, U/D(TXD) of XDCP to "0", count down
    for(i=0;i<8;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);     // set INC of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);     // set INC of XDCP to 1, generate a negative pulse on INC line
    }
        // set XDCP to step 120

    WrPortI(PGDR, &PGDRShadow, 0xaa);           // set PG7-PG0 = 10101010 (select even 24 DCPs, using AD5222)
    value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, value);           // set PF6-PF1=1, select all 6 hct373
    value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, value);           // set PF6-PF1=0, set 24 even CS of all 48 DCPs to 0

    BitWrPortI(PFDR, &PFDRShadow, 1, 0);         // set INC(CLKD) of XDCP to 1
    BitWrPortI(PCDR, &PCDRShadow, 1, 0);         // set PC0=1, U/D(TXD) of XDCP to "1", count up
    for(i=0;i<130;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);     // set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);     // set INC of XDCP to "1", generate a negative pulse on INC
line
    }
        // set XDCP to top
    BitWrPortI(PCDR, &PCDRShadow, 0, 0);         // set PC0=0, U/D(TXD) of XDCP to "0", count down
    for(i=0;i<8;i++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);     // set INC of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);     // set INC of XDCP to 1, generate a negative pulse on INC line
    }
        // set XDCP to step 120

    BitWrPortI(PDDR, &PDDRShadow, 0, 4);         // set PD4=0, turn off right laser
    BitWrPortI(PDDR, &PDDRShadow, 0, 5);         // set PD5=0, turn off left laser

while(1) {
    costate {
        waitFor( DelayMs(5L) );                 // wait for 5ms
        break;
    }
}
    BitWrPortI(PDDR, &PDDRShadow, 1, 4);         // set PD4=1, turn on right laser

```

```

    BitWrPortI(PDDR, &PDDRShadow, 1, 5);          // set PD5=1, turn on left laser

    Step_counter=120;

    for(i=0; i<48; i++) {
        XDCP_max[i] = 0;
        XDCP_min[i] = 0;
        XDCP_set[i] = 0;
        XDCPSetDone_max[i] = 0;
        XDCPSetDone_min[i] = 0;
    }

    for(i=0; i<12; i++)
        for(j=0; j<6; j++)
            samplingData[i][j] = 0;                // set 2D sampling Data array to 0

    intflag=0;

    while(1) {
        for(i=0; i<10; i++) {
            for(;;) {
                if(intflag==1) {
                    intflag=0;
                    break;
                }
            }
            samplingData[i][0]=testdata_left[0];    // store the sampling data of left side
            samplingData[i][1]=testdata_left[1];
            samplingData[i][2]=testdata_left[2];
            samplingData[i][3]=testdata_right[0];   // store the sampling data of right side
            samplingData[i][4]=testdata_right[1];
            samplingData[i][5]=testdata_right[2];
        }
        for(j=0; j<48; j++) {
            if(XDCPSetDone_max[j])
                continue;
            symbol=0;
            for(i=0; i<10; i++) {
                if((j>=0)&&(j<8)) {
                    if(bit(&samplingData[i][0],j-0)==0) {
                        symbol=1;
                        break;
                    }
                }
                else if((j>=8)&&(j<16)) {
                    if(bit(&samplingData[i][1],j-8)==0) {
                        symbol=1;
                        break;
                    }
                }
                else if((j>=16)&&(j<24)) {
                    if(bit(&samplingData[i][2],j-16)==0) {
                        symbol=1;
                        break;
                    }
                }
                else if((j>=24)&&(j<32)) {
                    if(bit(&samplingData[i][3],j-24)==0) {
                        symbol=1;
                        break;
                    }
                }
            }
        }
    }

```

```

else if((j>=32)&&(j<40)) {
    if(bit(&samplingData[i][4],j-32)==0) {
        symbol=1;
        break;
    }
}
else {
    if(bit(&samplingData[i][5],j-40)==0) {
        symbol=1;
        break;
    }
}
}
if(symbol==0) {
    XDCP_max[j]=Step_counter;
    XDCPSetDone_max[j] = 1;
}
} // set maximum steps for 48 DCPs

symbol=0;
for(i=0; i<48; i++) {
    if(XDCPSetDone_max[i]==0) {
        symbol=1;
        break;
    }
}

if((symbol==0)||((Step_counter<20)) // if output of DCP < 0.78V, break
    break;
else {
WrPortI(PGDR, &PGDRShadow, 0x00); // set PG7-PG0 = 00000000 (select odd 24 DCPs, using AD5222)
value = RdPortI(PFDR) | 0x7e;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
value = RdPortI(PFDR) & 0x81;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 odd CS of all 48 DCPs to 0
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); //set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); //set INC(CLKD) of XDCP to 1, generate a negtive pulse
on INC line

    WrPortI(PGDR, &PGDRShadow, 0xaa); // set PG7-PG0 = 10101010 (select even 24 DCPs, using
AD5222)
value = RdPortI(PFDR) | 0x7e;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
value = RdPortI(PFDR) & 0x81;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 even CS of all 48 DCPs to 0
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); //set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); //set INC(CLKD) of XDCP to 1, generate a negtive pulse
on INC line

    Step_counter--;
    intflag=0;
}
}

symbol=0;

WrPortI(PGDR, &PGDRShadow, 0x00); // set PG7-PG0 = 00000000 (select odd 24 DCPs, using AD5222)
value = RdPortI(PFDR) | 0x7e;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
value = RdPortI(PFDR) & 0x81;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 odd CS of all 48 DCPs to 0

```

```

BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC(CLKD) of XDCP to 1
BitWrPortI(PCDR, &PCDRShadow, 0, 0); // set PC0=0, U/D(TXD) of XDCP to "0", count down
for(i=0;i<130;i++) {
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); // set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC of XDCP to "1", generate a negative pulse on INC
line
} // set odd XDCPs to bottom
BitWrPortI(PCDR, &PCDRShadow, 1, 0); // set PC0=1, U/D(TXD) of XDCP to "1", count up
for(i=0;i<5;i++) {
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); // set INC of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC of XDCP to 1, generate a negative pulse on INC line
} // set XDCP to step 5, set XDCP to 0.2V

WrPortI(PGDR, &PGDRShadow, 0xaa); // set PG7-PG0 = 10101010 (select even 24 DCPs, using AD5222)
value = RdPortI(PFDR) | 0x7e;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
value = RdPortI(PFDR) & 0x81;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 even CS of all 48 DCPs to 0

BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC(CLKD) of XDCP to 1
BitWrPortI(PCDR, &PCDRShadow, 0, 0); // set PC0=0, U/D(TXD) of XDCP to "0", count down
for(i=0;i<130;i++) {
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); // set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC of XDCP to "1", generate a negative pulse on INC
line
} // set even XDCPs to bottom
BitWrPortI(PCDR, &PCDRShadow, 1, 0); // set PC0=1, U/D(TXD) of XDCP to "1", count up
for(i=0;i<5;i++) {
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); // set INC of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC of XDCP to 1, generate a negative pulse on INC line
} // set XDCP to step 5, set XDCP to 0.2V

intflag=0;
Step_counter=5;

while(1) {
    BitWrPortI(PDDR, &PDDRShadow, 1, 4); // set PD4=1, turn on right laser
    BitWrPortI(PDDR, &PDDRShadow, 1, 5); // set PD5=1, turn on left laser
    intflag=0;
for(;;) {
    if(intflag==1) {
        intflag=0;
        break;
    }
}
for(;;) {
    if(intflag==1) {
        intflag=0;
        break;
    }
}

BitWrPortI(PDDR, &PDDRShadow, 0, 4); // set PD4=0, turn off right laser
BitWrPortI(PDDR, &PDDRShadow, 0, 5); // set PD5=0, turn off left laser

for(i=0;i<12;i++) {
    for(;;) {
        if(intflag==1) {
            intflag=0;
            break;
        }
    }
}

```

```

}
samplingData[i][0]=testdata_left[0]; // store the sampling data of left side
samplingData[i][1]=testdata_left[1];
samplingData[i][2]=testdata_left[2];
samplingData[i][3]=testdata_right[0]; // store the sampling data of right side
samplingData[i][4]=testdata_right[1];
samplingData[i][5]=testdata_right[2];
}

for(j=0; j<48; j++) {
    if(XDCPSetDone_min[j])
        continue;
    symbol=0;
    for(i=0; i<12; i++) {
        if((j>=0)&&(j<8)) {
            if(bit(&samplingData[i][0],j-0)==1) {
                symbol=1;
                break;
            }
        }
        else if((j>=8)&&(j<16)) {
            if(bit(&samplingData[i][1],j-8)==1) {
                symbol=1;
                break;
            }
        }
        else if((j>=16)&&(j<24)) {
            if(bit(&samplingData[i][2],j-16)==1) {
                symbol=1;
                break;
            }
        }
        else if((j>=24)&&(j<32)) {
            if(bit(&samplingData[i][3],j-24)==1) {
                symbol=1;
                break;
            }
        }
        else if((j>=32)&&(j<40)) {
            if(bit(&samplingData[i][4],j-32)==1) {
                symbol=1;
                break;
            }
        }
        else {
            if(bit(&samplingData[i][5],j-40)==1) {
                symbol=1;
                break;
            }
        }
    }
    if(symbol==0) {
        XDCP_min[j]=Step_counter;
        XDCPSetDone_min[j] = 1;
    }
} // set minimum steps for 48 DCPs

if((symbol==0)||(Step_counter>30)) // if DCP output > 1.17V, break
    break;

else {
    WrPortI(PGDR, &PGDRShadow, 0x00); // set PG7-PG0 = 00000000 (select odd 24 DCPs, using AD5222)
    value = RdPortI(PFDR) | 0x7e;
}

```

```

    WrPortI(PFDR, &PFDRShadow, value);    // set PF6-PF1=1, select all 6 hct373
    value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, value);    // set PF6-PF1=0, set 24 odd CS of all 48 DCPs to 0
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); //set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); //set INC(CLKD) of XDCP to 1, generate a negative pulse
on INC line

        WrPortI(PGDR, &PGDRShadow, 0xaa); // set PG7-PG0 = 10101010 (select even 24 DCPs, using
AD5222)
        value = RdPortI(PFDR) | 0x7e;
        WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
        value = RdPortI(PFDR) & 0x81;
        WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 even CS of all 48 DCPs to 0
        BitWrPortI(PFDR, &PFDRShadow, 0, 0); //set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0); //set INC(CLKD) of XDCP to 1, generate a negative pulse
on INC line

        Step_counter++;
    }
}

for(i=0; i<48; i++) {
    if((XDCP_max[i]==0)||(XDCP_min[i]==0)) {
        XDCP_max[i]=128;
        XDCP_min[i]=128;
    }
    if((XDCPSetDone_max[i]==0)||(XDCPSetDone_min[i]==0)) {
        XDCP_max[i]=128;
        XDCP_min[i]=128;
    }
    if(XDCP_max[i]<XDCP_min[i]) {
        XDCP_max[i]=128;
        XDCP_min[i]=128;
    }
}

BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC(CLKD) of XDCP to 1
BitWrPortI(PCDR, &PCDRShadow, 0, 0); // set PC0=0, U/D(TXD) of XDCP to "0", count down

WrPortI(PGDR, &PGDRShadow, 0x00); // set PG7-PG0 = 00000000 (select odd 24 DCPs, using AD5222)
value = RdPortI(PFDR) | 0x7e;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
value = RdPortI(PFDR) & 0x81;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 odd CS of all 48 DCPs to 0
for(i=0; i<130; i++) {
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); // set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC of XDCP to "1", generate a negative pulse on INC
line
} // set odd XDCPs to bottom

WrPortI(PGDR, &PGDRShadow, 0xaa); // set PG7-PG0 = 10101010 (select even 24 DCPs, using AD5222)
value = RdPortI(PFDR) | 0x7e;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=1, select all 6 hct373
value = RdPortI(PFDR) & 0x81;
WrPortI(PFDR, &PFDRShadow, value); // set PF6-PF1=0, set 24 even CS of all 48 DCPs to 0
for(i=0; i<130; i++) {
    BitWrPortI(PFDR, &PFDRShadow, 0, 0); // set INC(CLKD) of XDCP to 0
    BitWrPortI(PFDR, &PFDRShadow, 1, 0); // set INC of XDCP to "1", generate a negative pulse on INC
line
} // set even XDCPs to bottom

```

```

    BitWrPortI(PCDR, &PCDRShadow, 1, 0);           //set PC0=1, U/D(TXD) of XDCP to "1", count up

    WrPortI(PGDR, &PGDRShadow, 0xff);             //set PG7-PG0 = 11111111
    //value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, RdPortI(PFDR) | 0x7e); //set PF6-PF1=1, select all 6 hct373
    //value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, RdPortI(PFDR) & 0x81); //set PF6-PF1=0, set all 48 CS of DCPs to 1

for(i=0; i<48; i++)
    XDCP_set[i]=( XDCP_max[i] + XDCP_min[i] )/2;    // calculate the settings of 48 DCPs

for(i=0; i<48; i++) {
    temp1=i/8+1; // get the bit number of Port F for DCP's CS
    temp2=i%8;   // get the bit number of Port G for DCP's CS
    if(temp2%2==0) {
        temp3=0x03;
        temp3<<=temp2;
    }
    else {
        temp3=0x01;
        temp3<<=temp2;
    }

    WrPortI(PGDR, &PGDRShadow, ~temp3);           //set PGi=0, or PGi&PGi+1 =0, select correct DCP for
    AD5222
    BitWrPortI(PFDR, &PFDRShadow, 1, temp1);      //set PFi=1(i=1-6), set CS of XDCPi to "0", other CS to
    "1"
    BitWrPortI(PFDR, &PFDRShadow, 0, temp1);
    for(j=0; j<XDCP_set[i]; j++) {
        BitWrPortI(PFDR, &PFDRShadow, 0, 0);      //set INC(CLKD) of XDCP to 0
        BitWrPortI(PFDR, &PFDRShadow, 1, 0);      //set INC(CLKD) of XDCP to 1, generate a negative
        pulse on INC line
    } //set XDCPi to the position

    WrPortI(PGDR, &PGDRShadow, 0xff);             //set PG7-PG0 = 11111111
    BitWrPortI(PFDR, &PFDRShadow, 1, temp1);      //set PFi=1
    BitWrPortI(PFDR, &PFDRShadow, 0, temp1);      //set PFi=0, set 1-8 CS of left side XDCP to "1"
}

    WrPortI(PGDR, &PGDRShadow, 0xff);             //set PG7-PG0 = 11111111
    //value = RdPortI(PFDR) | 0x7e;
    WrPortI(PFDR, &PFDRShadow, RdPortI(PFDR) | 0x7e); //set PF6-PF1=1, select all 6 hct373
    //value = RdPortI(PFDR) & 0x81;
    WrPortI(PFDR, &PFDRShadow, RdPortI(PFDR) & 0x81); //set PF6-PF1=0, set all 48 CS of DCPs to 1

    BitWrPortI(PDDR, &PDDRShadow, 1, 4);         // set PD4=1, turn on left laser
    BitWrPortI(PDDR, &PDDRShadow, 1, 5);         // set PD5=1, turn on right laser

    BitWrPortI(PBDR, &PBDRShadow, 1, 0);         // set PB0=1, turn off the LED indicator
}

void errorIndicate() {
    int i, syb;
    syb=0;
    for(; ) {
        if((BitRdPortI(PCDR, 1)==0)&&(syb==0))
            syb=1;
        else if((BitRdPortI(PCDR, 1)==1)&&(syb==1))
            break;
    }
}

```

```

costate{
    BitWrPortI(PBDR, &PBDRShadow, 0, 0);           // set PB0=0, turn on the LED indicator
    waitFor( DelayMs(50L) );                       // Turn on LED for 50ms
    BitWrPortI(PBDR, &PBDRShadow, 1, 0);           // set PB0=1, turn off the LED indicator
    waitFor( DelayMs(50L) );                       // Turn off LED for 50ms
}
}
}

```

## Source Code of TCP/IP communication in Linux

File name: server.c

```

#include <sys/socket.h>    /* socket definitions    */
#include <sys/types.h>    /* socket types        */
#include <arpa/inet.h>    /* inet (3) funtions   */
#include <sys/stat.h>

#include <unistd.h>       /* misc. UNIX functions */
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <time.h>
#include <signal.h>
#include <fcntl.h>

#include "control.h"

/* Global constants */
#define ECHO_PORT      (2002)
#define MAX_LINE      (1000)
#define LISTENQ        (1024) /* Backlog for listen() */
#define N_POINTS      (300)

struct msg_struct msg;

void timeout( int sig )
{
    printf("\nTimeout\n");
    exit(EXIT_FAILURE);
}

int main(int argc, char *argv[]) {
    int  list_s;    /* listening socket    */
    int  conn_s;   /* connection socket   */
    short int port; /* port number         */
    struct  sockaddr_in servaddr; /* socket address structure */
    unsigned char  buffer[N_POINTS*2];
    char  *endptr; /* for strtol()        */
    time_t time1;
    struct timeval tv_start, tv_end;
    int cc;

```

```

char buf;
struct sigaction sa;
int fd0, fd1, fd2, fd3;
unsigned char ret = 0;
//short int ret = 0;

/* Get port number from the command line, and
   set to default port if no arguments were supplied */
if ( argc == 2 ) {
    port = strtol(argv[1], &endptr, 0);
    if ( *endptr ) {
        fprintf(stderr, "ECHOSERV: Invalid port number.\n");
        exit(EXIT_FAILURE);
    }
}
else if ( argc < 2 ) {
    port = ECHO_PORT;
}
else {
    fprintf(stderr, "ECHOSERV: Invalid arguments.\n");
    exit(EXIT_FAILURE);
}

/* Create the listening socket */
if ( (list_s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
    fprintf(stderr, "ECHOSERV: Error creating listening socket.\n");
    exit(EXIT_FAILURE);
}

/* Set all bytes in socket address structure to
   zero, and fill in the relevant data members */
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(port);

/* Bind our socket addresses to the
   listening socket, and call listen() */
if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
    fprintf(stderr, "ECHOSERV: Error calling bind()\n");
    exit(EXIT_FAILURE);
}

/*
sa.sa_handler = timeout;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
sigaction(SIGALRM, &sa, NULL);
alarm(300); // Two minutes
*/

if ( listen(list_s, LISTENQ) < 0 ) {
    fprintf(stderr, "ECHOSERV: Error calling listen()\n");
}

```

```

    exit(EXIT_FAILURE);
}

/* Make a fifo for inter-process communications */
mkfifo("rtf0", 0777);
fd0 = open("rtf0", O_WRONLY, 0);
printf("done with rtf0\n");

mkfifo("rtf1", 0777);
fd1 = open("rtf1", O_WRONLY, 0);
printf("done with rtf1\n");

mkfifo("rtf2", 0777);
fd2 = open("rtf2", O_RDONLY, 0);
printf("done with rtf2\n");

mkfifo("rtf3", 0777);
fd3 = open("rtf3", O_WRONLY, 0);
printf("done with rtf3\n");

/* Enter an infinite loop to respond
   to client requests and echo input */
while ( 1 )
{
    printf("\n\n adafsedfasdfasdf\n\n ");
    read(fd2, &msg, sizeof(msg));
    if(msg.command == START_TASK)
    {
        printf("read START_TASK\n");
        break;
    }
}

int n = 0;
while ( 1 ) {

    /* Wait for a connection, then accept() it */
    if ( (conn_s = accept(list_s, NULL, NULL)) < 0 ) {
        fprintf(stderr, "ECHOSERV: Error calling accept()\n");
        exit(EXIT_FAILURE);
    }
    printf("\nSocket opened\n");

    /* Retrieve an input line from the connected socket
       then simply write it back to the same socket.  */
    int i = 0;
    gettimeofday(&tv_start, NULL);
    while( cc = read(conn_s, buffer, N_POINTS*2)){
        printf("read :%d\n", cc);
        //printf("%s\n", buffer);
        if( cc < 0 )
            break;
        gettimeofday(&tv_end, NULL);
        //printf("%s, %d\n", buffer, i);
        if(ret == 0)

```

```

    {
        if( (n = write(fd0, buffer, N_POINTS*2)) < 0)
        {
            fprintf(stderr, "WRITE: unable to write to fifo 0\n");
        }
        printf("r=%d\n", ret);
    }
    if(ret == 1)
    {
        if( (n = write(fd1, buffer, sizeof(buffer))) < 0)
        {
            fprintf(stderr, "WRITE: unable to write to fifo 1\n");
        }
        printf("r=%d\n", ret);
    }
    printf("write fifo=%d\n", n);

    if( (write(fd3, &ret, 1)) < 0)
    {
        fprintf(stderr, "WRITE: unable to write to fifo 3\n");
    }
    /*
    printf(" input strlen: %d", strlen(buffer));
    printf(" time: %d\n", abs(-tv_start.tv_usec+tv_end.tv_usec));
    */
    gettimeofday(&tv_start, NULL);
    memset(buffer,0,sizeof(buffer));
    i++;
    if(ret == 0)
        ret = 1;
    else if(ret == 1)
        ret = 0;
}

/* Close the connected socket */

if ( close(conn_s) < 0 ) {
    fprintf(stderr, "ECHOSERV: Error calling close()\n");
    exit(EXIT_FAILURE);
}
printf("\nSocket closed\n");

}

unlink("rtf0");
unlink("rtf1");
unlink("rtf2");
unlink("rtf3");
}

```