A Real-Time Optical Measurement System

by

Michael Anestis Colbert

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. A. H. Nayfeh, Co-chairman

Dr. J. C. McKeeman, Co-chairman

Dr. J. R. Armstrong

May, 1990

Blacksburg, Virginia

A Real-Time Optical Measurement System

by

Michael Anestis Colbert

Dr. A. H. Nayfeh, Co-chairman

Dr. J. C. McKeeman, Co-chairman

Electrical Engineering

(ABSTRACT)

Measurement of the dynamics of flexible structures is difficult because the motion is often complex and the structures are not well suited to the attachment of sensors. As a result, non-contact optical systems are used. However, optical systems produce large amounts of data which make their use in real-time measurement difficult. Conventional computers are not well suited to the processing requirements associated with data from optical systems. In this thesis, algorithms and architectures to reduce the data bandwidth of an optical measurement system are investigated. Simulations of the ability of the algorithms to find a target on a linear-array charge coupled device (CCD) camera are performed. The running maximum algorithm provides the best accuracy and speed and therefore is recommended.

A real-time architecture to implement the running maximum algorithm is developed. The architecture allows the optical system to operate at of 9700 frames/second. Experimental results from a prototype system show very good accuracy for both static and dynamic measurements.

Acknowledgements

I would like to thank my beloved wife, Linda, for her constant love, understanding, and encouragement. I am indebted to my parents for their love and to my brother, Costa, for his valuable insight and wisdom.

Gratitude is expressed to my advisor, Dr. Ali H. Nayfeh, for providing the opportunity to pursue graduate research, the support of this project and valuable advice and encouragement.

Deep appreciation is extended to my co-chairman, Dr. John C. McKeeman, for his advice, insight, and technical expertise. Without his help, friendship, and encouragement this thesis could not have been completed.

I would also like to thank Dr. James R. Armstrong for serving on my thesis committee.

I would like to thank my colleagues, Mr. Balakumar Balachandran, Mr. Marwan Bikdash, Mr. Tariq Nayfeh, Mr. Willman Remaklus, and Ms. Ginger Runyon for their valuable comments and suggestions.

Special thanks to Mr. Mahir Nayfeh for his friendship, suggestions, and advice throughout the course of my Master's degree and to Mr. Samir Nayfeh for his experimental expertise and patience during the testing of the optical system. I would also like to thank Taysir Nayfeh for providing the original concept of the measurement system and his comments during the development of the system. Without the expert help and friendship of Mrs. Sally G. Shrader, this manuscript could not have been completed.

This work was supported by the Office of Naval Research under Contract Nos. N0014-83-K-0184/NR 4322753 and N00014-90-J-1149.

Table of Contents

List of Illustrations

1.0 Introduction

The increased strength and lightweight properties of today's materials have prompted great interest in flexible structures. In particular, flexible structures have become increasingly important in the aerospace industry due to launch weight constraints. Flexible structures are an integral part of current aircraft designs, the proposed space station's solar arrays and slewing beams, and spaced-based antennas. As a result, the dynamics of flexible structures are being researched heavily.

Due to their increased use, the need for a complete understanding of the complex dynamics of flexible structures is critical. To gain this understanding, one requires precise measurements of their motion. Various devices are currently being utilized to obtain measurements of the motion of flexible structures. However, because of the nonlinear phenomena exhibited by flexible structures, such as the coupling of torsional motion with planar and non-planar motions, most contact measurement schemes are ineffective.

Non-contact, optical systems can measure the complex dynamics of flexible structures, but are severely limited by their signal and data processing systems. As a result, they cannot measure motions faster than 250 Hz. In this thesis, we develop a processing architecture which allows us to make accurate, high speed measurements of the motion of flexible structures with an optical measurement system.

1.1 Measurement Systems

Several systems for measuring the motion of flexible beams exist or are being developed. In choosing a system to measure the dynamics of flexible structures, one must consider several factors. Some measurement systems are only capable of measuring certain types of motion, while others cannot provide adequate sample rates. Further, the physical properties of some systems preclude them from being used in the measurement of flexible structures. Six systems which are currently being used are outlined in this section to illustrate their limitations and to determine which system could be improved to yield better measurements.

1.1.1 Strain Gauges

One method used for measuring the dynamics of flexible structures in the laboratory is based on attaching strain gauges to the structure. By using a strain gauge mounted on the beam as one arm of a Wheatstone bridge, one can measure the displacement of the beam. As the beam bends, its strain increases and as a result, the resistance of the strain gauge changes. This change in resistance causes the bridge to become unbalanced, resulting in an output voltage proportional to the strain on the displaced beam. This method can be used fairly accurately to measure small displacements if the strain gauge is mounted correctly and its lead wires are not permitted to increase the stiffness of the structure [1]. The accuracy of the strain gauge, however, is based on the underlying assumptions that a static calibration of the bridge will be correct under dynamic conditions and that the strain on the beam is a linear function of its displacement. If either of these assumptions breaks down, as the latter does under large displacements, the measurement system fails. Additionally, with these types of systems, measurement of out-of-plane motions and torsion are very difficult. The accuracy of the measurement of these phenomena depends on an *a priori* knowledge of the beam's motion so that the strain gages can be positioned correctly. If the strain gauge is not mounted so that motion in only one plane affects it, the motion in several planes may become coupled, which may result in a loss of accuracy. While measuring the dynamics of a graphite/epoxy composite beam structure, Balachandran [2] noted that one of the flexural frequencies of the structure

was approximately equal to one of its torsional frequencies. Both the flexural and torsional modes were clearly observable with the naked eye, however, the frequency response of the structure obtained using strain gauges and a spectrum analyzer could not distinguish between them [2].

1.1.2 Accelerometers

Accelerometers are also used for motion measurement. An accelerometer contains piezoelectric elements which act as springs connecting the base of the accelerometer to a seismic mass. When the accelerometer is moved, a force, proportional to the product of the acceleration and mass of the seismic mass, acts on the piezoelectric elements. The elements produce a charge proportional to the response force. Since the seismic mass is constant, the charge produced is proportional to the acceleration of the seismic mass. Because it accelerates with the same magnitude and phase as the base the output of the accelerometer is proportional to the acceleration of the base. These characteristics allow accelerometers to measure the acceleration of the surface to which the base is mounted. Once the acceleration is known, the displacement can be found by integrating the signal twice [3].

Using accelerometers to measure the motion of lightweight flexible structures, however, has serious limitations. The most serious of these is the accelerometer's size. The mass of the accelerometer precludes it from being used on lightweight structures, as mounting the accelerometer changes the

dynamics of the structure. Moreover, the error in the measurement of the base acceleration increases rapidly as the measured frequency approaches the transducer's lowest resonant frequency. In addition, it is not possible to mount the accelerometer on some elastic elements such as a string. The physical properties are not the accelerometer's only source of limitation, though. The signal produced by the accelerometer is proportional to the acceleration. As mentioned before, the signal must be integrated twice to obtain the displacement. Since there are no ideal integrators, the signal obtained after two integrations may contain errors. Other sources of error include the accelerometer's sensitivity to temperature changes because of the piezoelectric elements, and its sensitivity to proper mounting and orientation due to the geometries of the seismic masses.

1.1.3 Optical Sensing

The most attractive systems for measuring the motion of flexible structures are optical ones as they have many advantages over the ones previously mentioned. Optical systems can provide non-contact measurements of the motion of structures with geometries unsuited for attachment of sensors and accurate measurements of complicated motions. However, imaging systems produce large amounts of data, thereby requiring large memories and powerful computing architectures [4]. Several systems have been proposed

using both holographic imaging and charge coupled devices (CCDs). A brief description of these systems is given in the following sections.

1.1.3.1 Holography

Currently under development at NASA Langley Research Center is an optical measurement system which uses holographic memories and optical processing techniques to provide motion measurement and control of vibrations in a cantilever beam [5]. In this system, the beam whose motion is to be measured is illuminated with a multi-frequency laser light of known spectral distribution. This light is collected with collimating and focusing optics. At the same time, a reference wave is made to interfere with the focused wave causing an interference pattern which can be saved in an optical memory. The hologram is read out of memory with a second reference wave. This results in the reconstruction of the original image reflected off the structure. Using this setup, one produces a contour map of the structure over time, which can be used to measure the changes in the contour of the structure and thus its motion [5].

This system produces a large quantity of optical information. If the data is converted to a digital form, standard computers cannot process the data rapidly enough. As a result, optical processing is mandatory.

1.1.3.2 CCD Arrays

One device used in optical sensing is the charge coupled device (CCD). CCDs are semiconductor components which perform the general functions of image sensing, analog signal processing, and digital or analog memories. As imaging devices, CCDs are configured as either two-dimensional array or linear-array devices. In either configuration, each individual element in the array is called a pixel, and the output of an entire array of pixels is called a frame. The scan or frame rate is generally defined as the number of frames which can be acquired by the camera per second.

CCD cameras (e.g., video cameras) have been used to capture motion for some time. However, because the scan rate of two-dimensional arrays is slow (only 120 frames/sec for a 256x256 element array [6]), their use in motion measurement has been very limited. Linear-array cameras provide the scan rate (up to 9700 frames/sec for a 2048x1 element array) necessary for measuring the motion of flexible structures but only along a straight line. To utilize linear-array cameras, one needs to employ a method for capturing the motion in all planes.

One configuration of linear-array CCDs for measuring the velocity of moving objects was proposed by Bhuiya and Farhat [7]. Their system consists of four CCD arrays with threshold detectors at their bases configured in a cross. A fifth threshold detector placed at the center of the cross is used as a reference,

as shown in figure 1. The operation of this configuration is as follows. If the center threshold detector senses a change in the image signal (i.e., the image focused on the detectors by a lens), it activates the four threshold detectors. Once activated, these threshold detectors initiate a scan on their respective CCD arrays if they detect a change in the image signal. Using this configuration, one can calculate the object's velocity from the number of pixels, on the branch which is activated by a change of light intensity on its threshold detector, which have been crossed. Reading only the activated array significantly reduces the amount of data to be processed. This system, however, has some restrictions which make it inadequate for measuring the motion of a flexible beam. In order for this system to work, the object must cover an adequate area of the array. For measurement of the beam's motion, though, we desire to detect only a few points on the beam. For 3-D motion measurements, this system also requires that the image excites elements on at least three of the arrays. For complicated motions, this requirement may or may not be met. Unfortunately, as this system is only a theoretical model, no measurements have actually been made using it. It is, therefore, only speculation as to whether it could effectively perform three-dimensional motion measurements.

In another CCD array imaging system, Kotnik, Yurkovich, and Ozguner [8] and Ozguner, Yurkovich, Martin and Kotnik [9] used an incandescent lamp mounted to the tip of a flexible beam and a linear-array CCD camera to detect the tip's position. A 50 Hz to 500 Hz scan rate is used and the system is

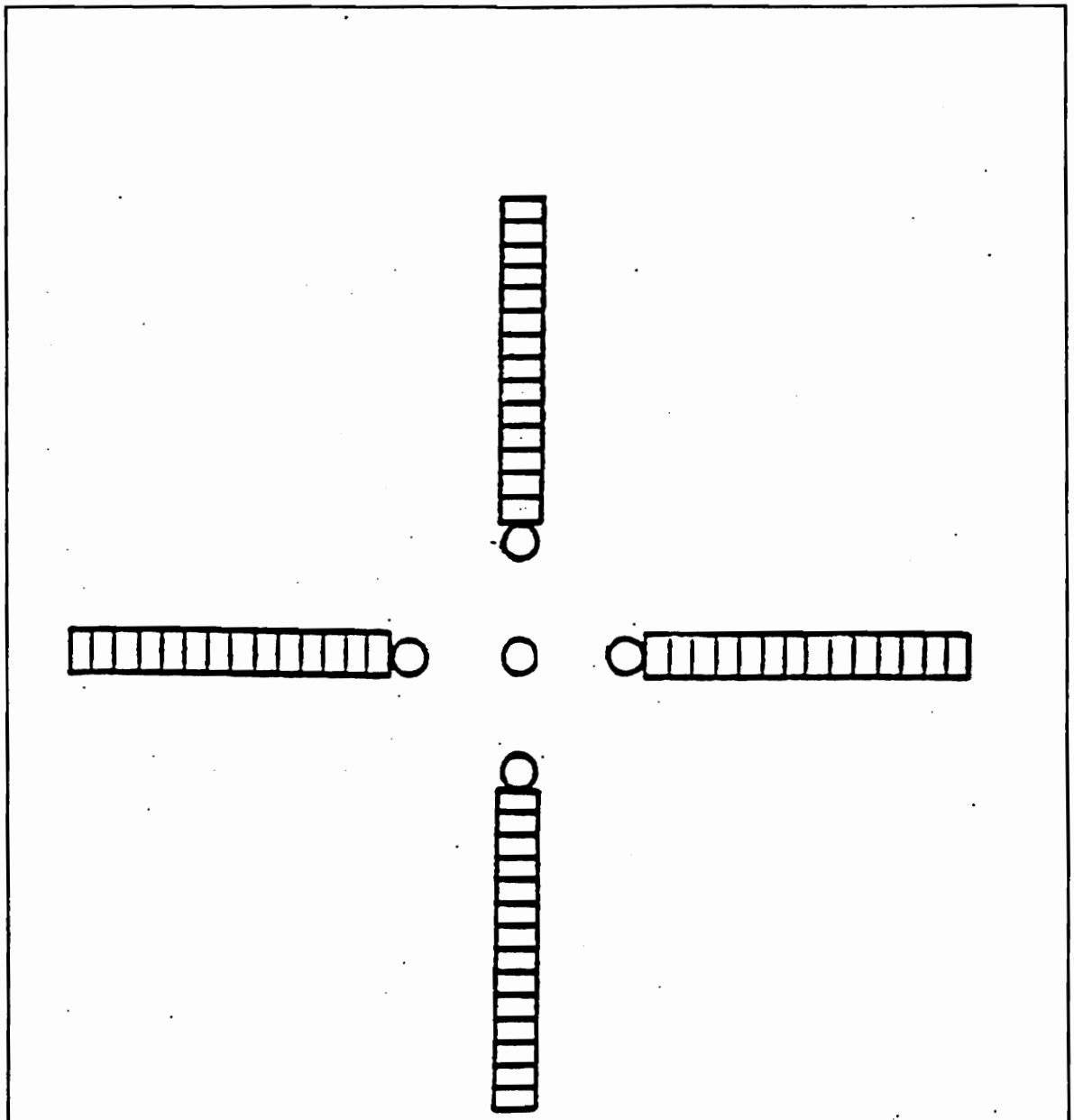


Figure 1. Bhuiya and Farhat's CCD configuration for measuring motion.

capable of resolving a horizontal displacement of the one meter structure to approximately 2 mm. The data from the 512 element linear-array camera is collected by a Digital Equipment Corporation MicroVax II running at 20 MHz. This configuration, however, can only measure motion in one plane, the plane along the array's length. Any motion out of the array's plane, or field of view, cannot be measured. If the structure moves out of the camera's field of view the image completely misses the array. This condition results not only in loss of motion measurement in planes out of the array's field of view but also along the plane of the array as well.

Seymour [10] developed a flight detection measurement system (FDMS) to provide accurate measurement of the position of a point in space with respect to a reference point. Specifically, the system was designed to measure aircraft wing deflections during flight. The system is similar to the one used by Ozguner, *et al.*, however, the FDMS system employs a cylindrical lens in front of the CCD array to stigmatize the light rays from remote light emitting diodes (LEDs) to lines on the CCD array. The use of the cylindrical lens eliminates the data loss which occurs when the remote target moves out of the linear-array's field of view. The output of the CCD array contains a voltage spike representing the position of the target with respect to the frame. The data is shifted out of the CCD camera and the output is processed sequentially to determine the pixels that are excited by the stigmatized line. The result is the position from the target point. The processing consists only of run-length encoding and/or temporal differencing. This system yields quantitative data

from the 512 pixel CCD array at a maximum rate of 200 Hz. Multiple targets can be tracked by time multiplexing [10,11]. Tests performed by Seymour [10] and Angelis [11] indicate that the FDMS provides an accuracy equal to or better than that obtained by either strain-gage or accelerometer systems [10,12].

Seymour's system provides the ability to measure the motion of a structure without affecting its dynamics [11]. However, the scan rate, 200 Hz, only allows measurement of motions with frequency components less than 100 Hz. The maximum scan rate of the CCD cameras is much higher than the frame rates achieved by the Seymour system. The factor limiting the sample rate is the processing system being employed. Because Seymour's system can only process an input data stream of 100 Kbytes/sec [11], it must reduce the frame rate to avoid saturating the bandwidth of the system.

1.2 Processing Systems

It is well known that standard complex instruction set computer (CISC) architectures are not well matched to the required image processing where large amounts of data must be processed within a limited time. Fehevari, Lambrechts, Bactens, and Oosterlink [13] state that there are two techniques for achieving the speed and power needed to process image data. One can either use p general purpose processors and split the data into p parts or

develop a special hardware architecture optimized for fast execution of an algorithm. The following systems illustrate the effectiveness of both techniques.

A system that processes a raw data stream, similar to the one produced by the CCD cameras in the FDMS, was used by Van den Hende and Constandse [14]. The raw data, contains mostly background levels and few peaks which represent events. The positions and magnitudes of these events must be saved and the irrelevant background data can be discarded. The system employs two 16-bit processors, constructed with AM 2900 family bit-slice processors. These processors reduce the data needed to store the results of mass spectrometric experiments. The sampling rate of the mass spectrometer is 400 KHz with 16-bits of resolution. This corresponds to an information bandwidth of 800 Kbytes/sec. The design of their data processing unit can reduce the information bandwidth to 3 Kbytes/sec. At this rate, the data can be stored in the secondary memory device of standard computer systems [14].

Bermbach, Scharfenburg, and Gambel [15] used a single custom processor optimized to run-length encode the raw data generated by a line scan measurement system. The coding provides binarization and edge detection, reducing the data rate to a level which can be handled by a standard computer. Wilkinson, Atkins, and Rogers [16] demonstrated that the data from high I/O bandwidth systems, such as a Positron Emission Tomograph (PET),

can be managed with a custom parallel processing system made up of several Motorola 68020 processors.

1.3 Thesis Overview

Because of the benefits of non-contact measurements, accurate measurements of complex dynamics, and measurements of motions of structures with geometries unsuited for sensor attachment, optical sensing is the method chosen for implementation in this thesis. Specifically, a variation of the FDMS measurement system developed by Seymour is implemented. The processing systems developed by Van den Hende, *et al.*, Bermbach, *et al.*, and Wilkinson, *et al.* reduce the raw data rate of their respective systems to a much greater degree than achieved by the processing system in the FDMS. These systems demonstrate the ability of achieving very high data bandwidth reduction using small customized architectures. In this thesis, a real-time processing system for reducing the data bandwidth of the FDMS is developed. The basic measurement system developed by Seymour coupled with the custom control and processing system developed in this thesis provides very high speed (9700 frames/sec), accurate measurements of the motions of flexible structures. The frame rate of the system developed in this thesis is 45 times that of the FDMS, thus allowing us to measure motions with frequency components up to approximately 4.5 KHz.

The organization of this thesis is as follows. In Chapter Two, we describe in detail the basic operation and components of the measurement system. As mentioned earlier, optical systems produce large amounts of data. In Chapter Three, we compare algorithms capable of processing this data as well as the computing requirements and architectures needed to implement these algorithms. A selection of the algorithm and architecture to be implemented in this thesis is made in Chapter Three. In Chapter Four, we describe in detail the hardware implementation of the algorithm and architecture chosen in Chapter Three. Chapter Four also describes the data acquisition software developed for this system. Measurements made with the system are presented and discussed in Chapter Five. Chapter Six presents conclusions and recommendations for further research.

2.0 Measurement System Description

2.1 Objective

The objective of the system developed in this thesis is to provide accurate measurements of the motions of flexible structures without the limitations associated with the measurement systems described in Chapter One.

The measurement system must not affect the structure's motion. Conversely, the structure's motion should not have an effect on the accuracy of the measurement system. The FDMS meets both these requirements [10-12] and is used as the basis for our high speed processing system.

2.2 Description

A basic one-dimensional measurement system consists of a cylindrical lens, a target light source, a CCD linear-array camera, and a processing system, as shown in figure 2. The operation of the measurement system is as follows. The scene containing the structure to be measured is stigmatized to a line image by a cylindrical lens. The line image of the scene is then captured by the linear-array CCD camera. The CCD array converts this line image of the scene to a discretized analog signal. Because a target light source is attached to the structure, the scene contains a bright spot corresponding to the target's position with respect to the array [11]. The processing system determines this position from the line image generated by the CCD camera. Detailed descriptions of the first three components in our system are given in the following sections. The details of the processing system are presented in Chapters Three and Four.

2.2.1 CCD Cameras

The system described in this thesis uses a charge coupled device (CCD), linear-array camera to sense the motion of the structure. Specifically, a Fairchild CCD1500R camera containing a CCD line-scan image sensor is employed. The CCD sensor is a monolithic component containing a single row of 2048 pixels, two analog-transport shift registers, and two output-sense

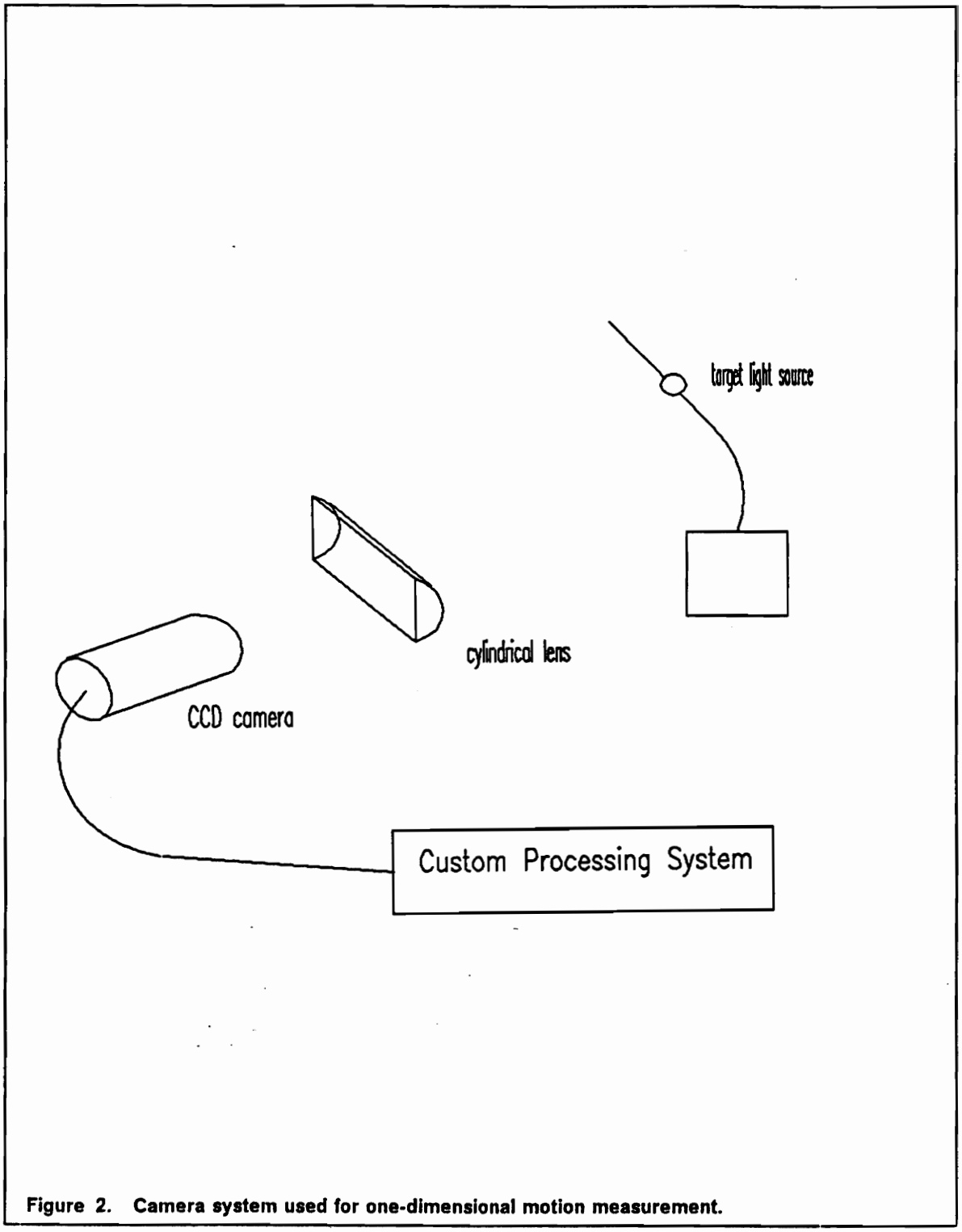


Figure 2. Camera system used for one-dimensional motion measurement.

amplifiers, as shown in figure 3. Light energy falling on the pixels generates electron charge packets which are proportional to the product of the exposure time and the incident light intensity. The exposure time is the reciprocal of the frame rate at which the camera is operating. The pixel charge packets are transferred in parallel to two analog-transport registers in response to the frame-rate clock input from the camera. The transport registers, in response to the data-rate clock, deliver the packets in sequence to two integrating charge sense amplifiers, where they are converted to proportional video signal voltage levels [17]. Thus, each time the camera is scanned, an analog signal representing the spatial distribution of light incident on its array of pixels is produced. This output signal varies from a thermally generated noise background of 0.2 volts for zero illumination to a one volt maximum when the array is saturated by bright illumination. The maximum difference in the response levels between the most and least sensitive elements under uniform illumination is 100 mV. The photosensitive pixels are 13 microns by 13 microns square. There is a dead space between pixels which is not photosensitive. This space is approximately five microns. Because of the dead space, the possibility of the target falling between two pixels exists. However, in this thesis, the actual target is assumed not to be a point source, because at the close distances involved, it covers several pixels on the array. The pixel adjacent to the dead space where the target light falls generates the greatest charge.

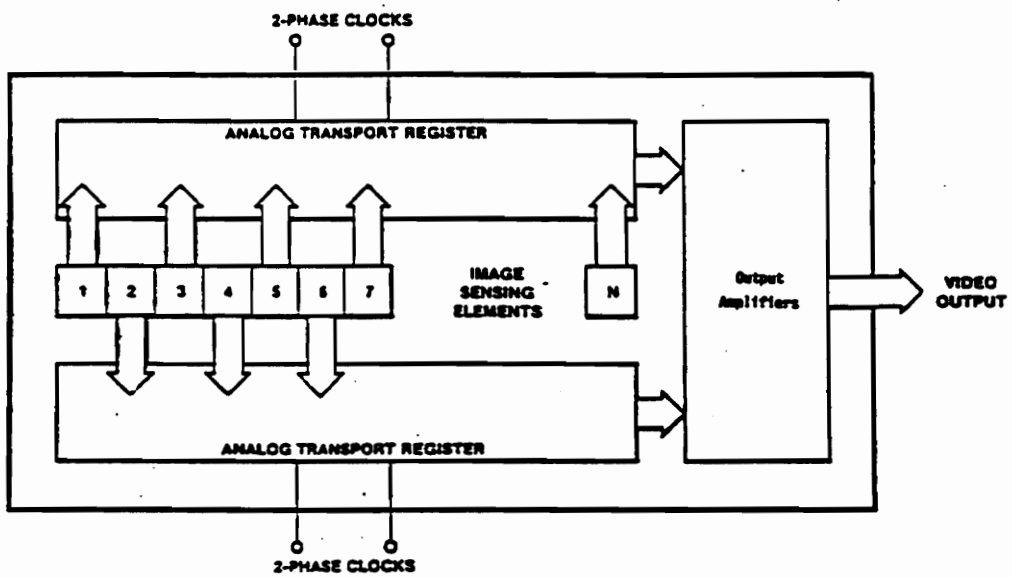


Figure 3. Internal configuration of CCD linear array. (Fairchild)

A CB1000R camera control unit provides the necessary power supply voltages to the CCD camera. The CB1000R also buffers the pixel data-rate clock and the frame-rate clock, generated by the control and processing system (CPS). The pixel data-rate clock and the frame-rate clock are unbalanced differential lines which drive the camera to improve noise immunity. In addition, the CB1000R generates a START-OF-FRAME pulse at the beginning of each frame. This active low pulse lasts two periods of the pixel data-rate clock. The control unit also generates a VALID VIDEO signal which is active high. The VALID VIDEO signal transitions from low to high synchronously with the START-OF-FRAME pulse, however, unlike the START-OF-FRAME pulse, it remains high for the duration of a frame. A diagram of the CB1000R's timing relationships is given in figure 4.

The maximum data-rate clock at which the CCD1500R will operate correctly is 20 MHz. The minimum time to shift a 2048 pixel frame out of the camera is then,

$$\text{Frame Period} = 2048 \text{ pixels} \times \frac{1 \text{ sec}}{20 \text{ Mpixels}} = 102.4 \text{ microseconds} \qquad (1)$$

Because a new frame is being collected by the CCD sensor as the previous frame is being shifted out along the transport registers, a new scan cannot be initiated until the previous frame is completely shifted out of the camera. If this requirement is not met, data from the two frames will be mixed. The corresponding frame-rate is, therefore,

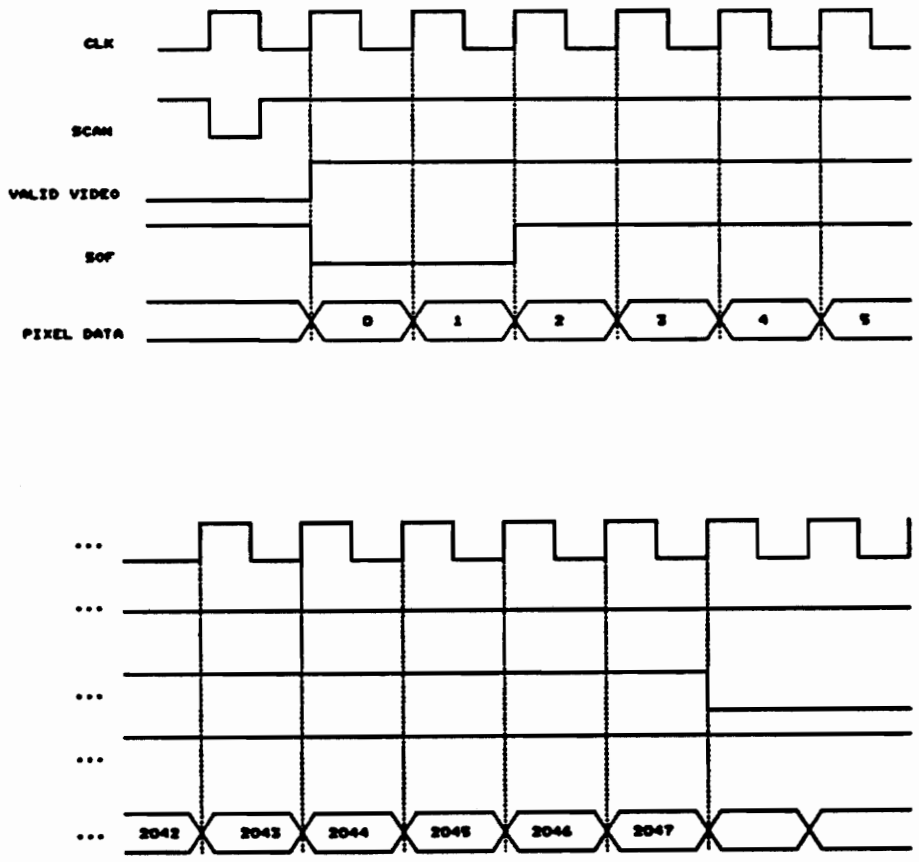


Figure 4. Timing diagram of CB1000R output signals.

$$\text{Frame Rate} = \frac{1}{\text{Frame Period}} \quad (2)$$

2.2.2 Cylindrical Lens

Because the camera contains a linear CCD array, only motion in the plane of the array can be measured. If the motion continues out of the plane of the array, the motion cannot be detected. This situation is rectified by using a cylindrical lens to focus the target light source onto a line in the plane of the array. The geometry of a cylindrical lens compresses or "squeezes" a plane onto a line, as shown in figure 5. Any point in front of the cylindrical lens is focused onto a line behind it. Thus, the cylindrical lens and linear-array CCD combination is sensitive only to motion in the plane parallel to the length of the CCD array.

2.2.3 Target-Light Source

A great deal of knowledge about the dynamics of a flexible structure can be inferred by tracking one of its points. The single point, or target, must be illuminated in order to register on the CCD array. The target can be an attached light source, such as a micro-incandescent bulb, laser diode, light emitting diode (LED), or fiber-optic bundle. The target can also be a reflector such as a retro-reflective tape, which directs illumination back to its source, a

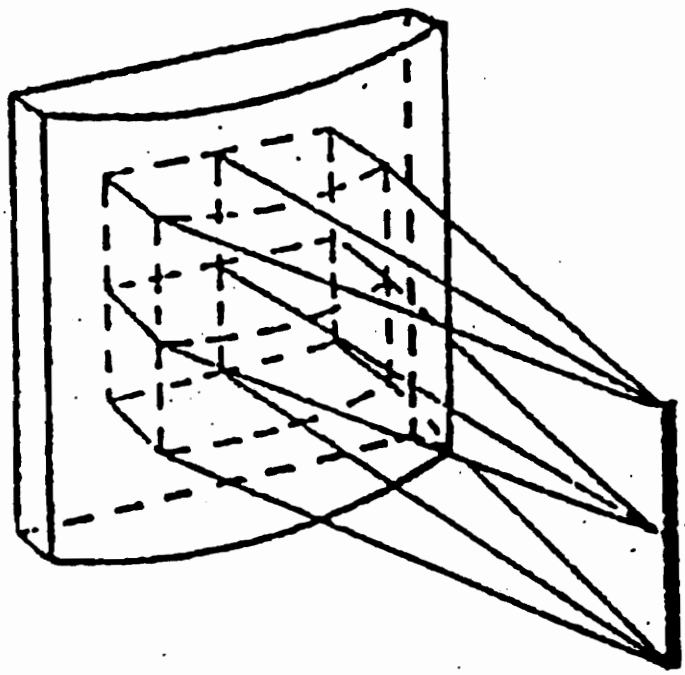


Figure 5. Principle of operation for a cylindrical lens. (Ealing Optics)

reflective paint, or a mirror. If the positions of more than one point on the structure are to be measured, several light sources can be attached to the structure and multiplexed in time.

2.3 *Positioning*

Because the cylindrical lens and linear-array are sensitive only to motions in planes parallel to the linear-array, one CCD array and cylindrical lens constitute a single-axis displacement sensor. Two camera-lens pairs provide a two-axis detector. Three-dimensional motion measurements can be made by positioning three camera-lens pairs around the structure such that each camera measures motion in one plane. Once the directional data are acquired by the processing system, the displacement of the target can be calculated. The following section discusses the algorithm needed to calculate the displacement from the data returned by the measurement system.

2.3.1 Position Algorithm

With a three-axis detector, three mutually orthogonal planes of motion are measured. The intersection of these three planes yields a point. This intersection is found as follows. Referring to figure 6, we let

L_n be the distance from the cylindrical lens to the target,

θ_n be the angle from the center of the array to the target, and

k be an empirical constant.

Then,

$$\tan k\theta_1 = \frac{y}{L_1 - x} \tag{3}$$

$$\tan k\theta_2 = \frac{x}{L_2 - y} \tag{4}$$

$$\tan k\theta_3 = \frac{z}{L_3 + y} \tag{5}$$

Solving for the displacement y yields

$$y = \frac{L_1 \tan k\theta_1 - L_2 \tan k\theta_2 \tan k\theta_1}{1 - \tan k\theta_2 \tan k\theta_1} = \frac{L_1 - L_2 \tan k\theta_2}{\cot k\theta_1 - \tan k\theta_2} \tag{6}$$

Then, it follows from (4) and (5) that the x and z displacements are given by

$$x = (L_2 - y) \tan k\theta_1 \tag{7}$$

$$z = (L_3 + y) \tan k\theta_3 \tag{8}$$

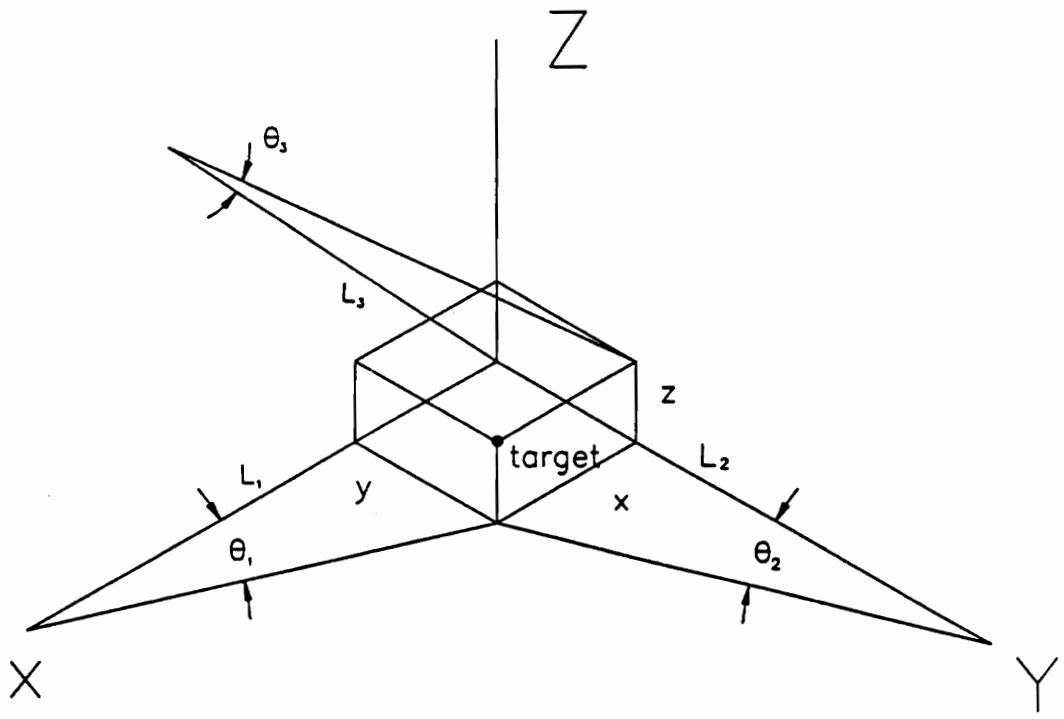


Figure 6. Geometric relationship between the target and the measurement system.

2.3.2 Single-Axis Detector

As mentioned above, when one camera-lens pair is employed, the system provides measurement of motion along one axis. With a single-axis detector, the angular displacement data can be translated directly into the displacement of the structure. The following relation describes the resolution of the system using direct calibration:

$$\text{Minimum Resolution} = \frac{\text{maximum displacement}}{\text{number of pixels}} \tag{9}$$

For example, if the maximum displacement of the target-light source on the structure is 100 mm, the resolution is

$$\frac{100 \text{ mm}}{2048 \text{ pixels}} = 49 \text{ microns} \tag{10}$$

3.0 Signal Preprocessing Algorithms

The measurement system described in Chapter 2 yields an analog signal, or frame, representing the spatial distribution of light in the scene being measured. Although the signal is analog, it is made up of discrete pixels shifted out of the camera. A processing algorithm must be developed to analyze the signal, or pixel stream, generated by the camera.

Both Fathi and Fines [18] and Pfister [19] characterize processing of signals generated by imaging systems by high sample rates and heavy computational loads. The processing can sometimes be partitioned into preprocessing and post-processing operations. The former provides a coarse set of results and reduces the data bandwidth. The latter performs higher level operations yielding a smaller and more accurate set of results. The custom control and processing system (CPS) developed in this thesis is partitioned to perform the preprocessing tasks. The host personal computer, running the custom data acquisition software described in Chapter 4, performs the post-processing

tasks. The algorithms which perform the preprocessing can be described as high-speed algorithms whose performance is closely matched to the physical limits of the sensor [19].

The purposes of the preprocessing algorithm are two-fold: (a) to determine the position of maximum energy within each frame and (b) to reduce the serial pixel stream generated by the camera to an address stream representing the target's position on the array. If the second task can be accomplished in real-time, only one address per frame, rather than 2048 pixels per frame, is transferred to the personal computer for post-processing. This greatly alleviates the amount of required post-processing. The available execution time is a major factor in determining the level of complexity of the preprocessing algorithm. In this chapter, we compare three algorithms capable of performing the specified preprocessing tasks. They are run-length encoding (RLE), running maximum (RM), and automatic background compensation (ABC). The accuracy and speed as well as ease of implementation of these three algorithms are compared. A digital-frame grabber was built to digitize and capture individual frames so that the performance of these algorithms can be compared using realistic simulations. An example of a captured frame is shown in figure 7.

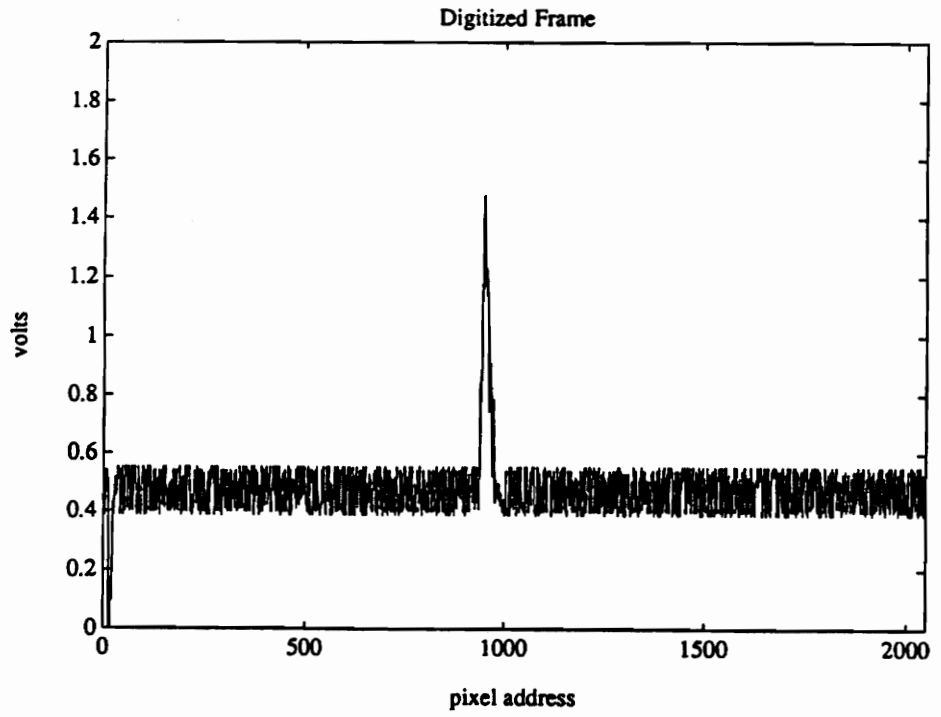


Figure 7. Example of a digitized frame.

3.1 Description of Algorithms

Hansen and Elyashar [20] noted that there are two primary types of noise associated with optical detectors: clutter or reflections, which can be mistaken as targets, and high-frequency electronic noise generated by the detector itself. The preprocessing algorithm must be able to accurately detect the target signal in the presence of either of these types of noise.

Since the camera output signal is analog, signal processing can be accomplished either in the analog or digital domain. The three algorithms, RLE, RM, and ABC, can be implemented in either the analog or digital domain. However, for the purpose of comparing the accuracy of the algorithms we will use digital implementations.

3.1.1 Run-Length Encoding

Run-length encoding [10,11,12,17,20] is used to reduce the data bandwidth of the incoming signal to a level manageable by a standard computer. To initiate the RLE algorithm, the user chooses a voltage threshold above the expected background level [12]. Then, a pixel voltage above the threshold is defined as light and a pixel voltage below the threshold is defined as dark. A transition location is a position in a frame where the voltage level changes from either dark to light or light to dark. The RLE algorithm reduces the raw

pixel data rate by converting a frame of pixel voltages to a frame of transition locations, which is used to determine the target location. Referring to the flowchart in figure 8, we describe the operation of the RLE algorithm as follows. First, the current level needed by the algorithm is initialized to dark. The algorithm then initiates a scan and compares each pixel, shifted sequentially out of the CCD array, to the threshold. If the pixel is above the threshold and the current level is dark, the address of the pixel is saved and the current level is set to light. If the pixel is below the threshold and the current level is light, the address of the pixel is saved and the current level is set to dark. If neither of these conditions are true, no action is taken. The next pixel shifted out of the CCD array is then checked for the same conditions. This operation is done for each of the 2048 pixels in the frame as they are shifted out of the camera. When the frame is complete, the target position is determined from the transition locations saved. The C language implementation of this algorithm is given in figure 9.

A typical run-length encoded frame would yield two transition locations: the transition location from dark to light and the transition location from light to dark. The average of the two locations yields the target's position. If multiple transition regions are found the algorithm must determine the region that contains the target. This entails extrapolating the target's possible position from previous frames and choosing the transition locations which fall nearest to the expected position [20].

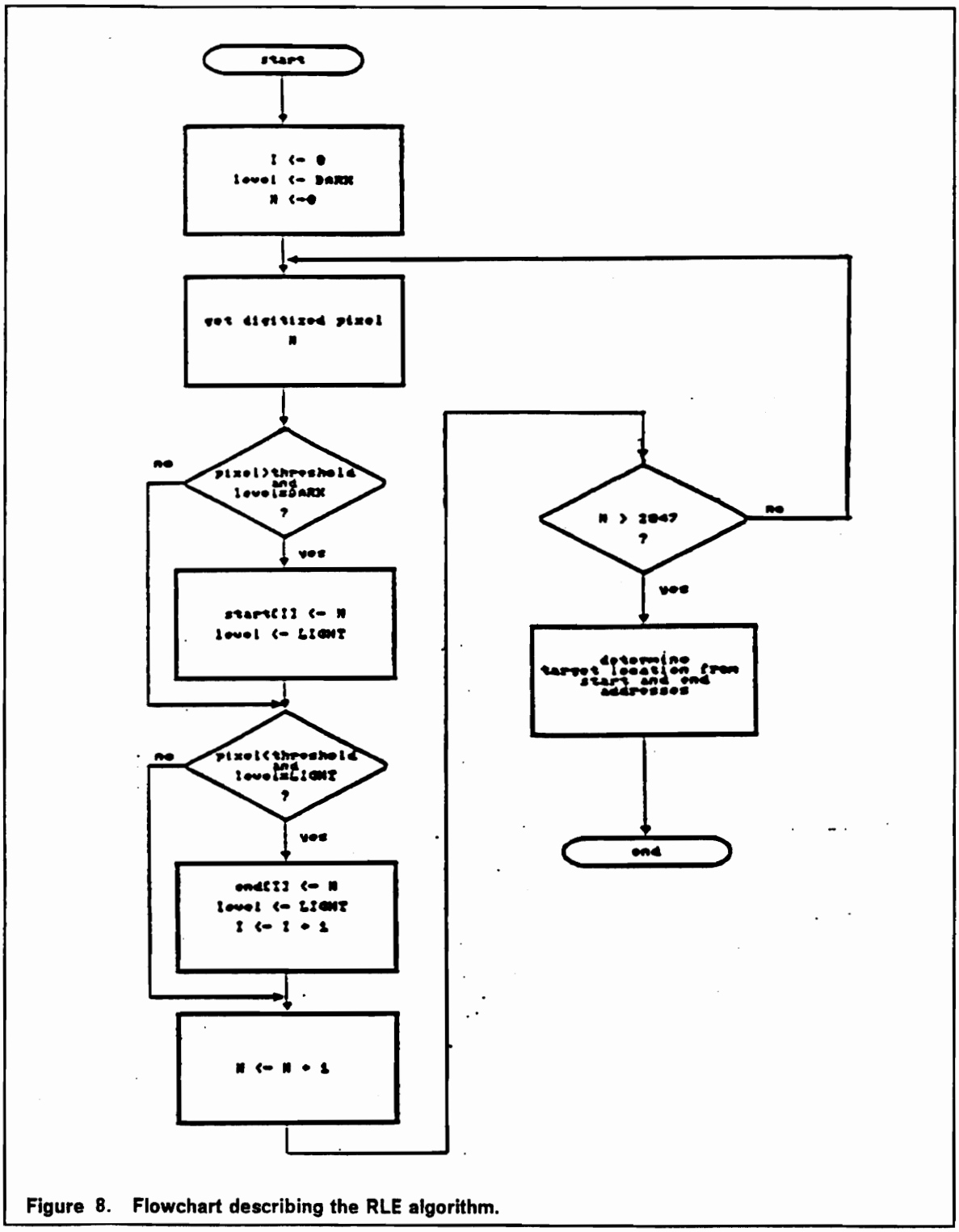


Figure 8. Flowchart describing the RLE algorithm.

```
void RLE(float threshold int frame,int polarity, int start, int end)
{
  int i,N,level;

  i = 0;
  level = 0;
  for(N = 0;N<2048;N++){
    if( (frame[N] > threshold) && (level == 0)){
     start[i] = N;
     level = 1;
     i++;
    }
    if( (frame[N] < threshold) && (level == 1)){
      end[i] = N;
      current_level = 0;
      i++;
    }
  }
}
```

Figure 9. C language code for the RLE function.

3.1.2 Running Maximum

The RM algorithm is a variation of the peak algorithm proposed by Hansen and Elyashar [20]. Their proposed algorithm sets a threshold based on an estimate of the selected target's intensity, rather than on an estimate of noise. The RM algorithm finds the target directly from the raw pixel data as it is shifted out of the CCD array without using a predetermined threshold. The RM algorithm searches the frame sequentially for the pixel with the maximum value and stores its address as the target's location. Referring to the flowchart in figure 10, we describe the operation of the RM algorithm as follows. The current maximum is set equal to zero. A scan is initiated and the first pixel value is compared to the current maximum. If the current pixel value is greater than the current maximum, its value is saved as the new current maximum and its address is saved as the target's location. The next pixel shifted sequentially out of the CCD array is then compared to the current maximum. Again, if the pixel is greater than the current maximum, the current maximum is set equal to the pixel value and the address is stored as the target's location. This operation is performed on each pixel as it is shifted out of the CCD array. When the entire frame of 2048 pixels has been shifted out of the array, the RM algorithm returns the address of the maximum pixel. Figure 11 shows the C language implementation of this algorithm.

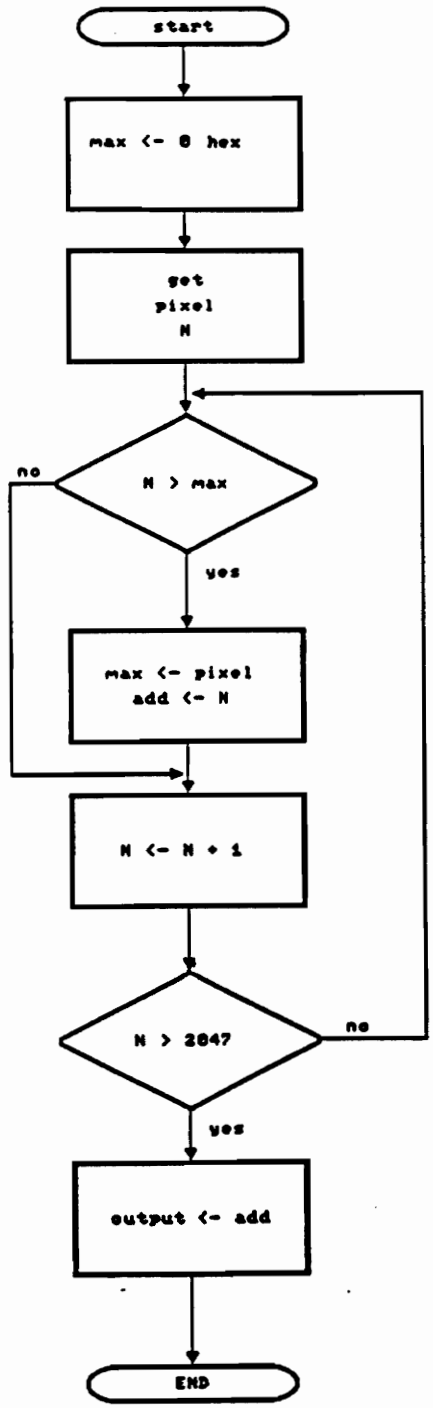


Figure 10. Flowchart detailing the RM algorithm.

```
int RM(int frame)
{
 int maximum, N;

  maximum = 0xff;
  for(N=0;N<2048;N++){
   if(frame[N] > maximum){
     maximum = frame[N];
     add = N;
   }
  }
  return add;
 }
```

Figure 11. C language code which implements the RM algorithm.

3.1.3 Automatic Background Compensation

Automatic background compensation (ABC) [11,12,20] is used to improve the accuracy of the measurement made with the Seymour FDMS. The automatic background compensation algorithm operates in the same manner as the run-length encoding, however, it eliminates the background noise from each frame. The ABC algorithm requires two sequential frames to produce a single output frame. First, a frame of data of the scene without the target light source is stored in memory. A second frame of the same scene with the target light source illuminated is then taken. The temporal difference of the two yields a frame representing only the target data. The temporal differencing acts as a high-pass filter, allowing only objects with high spatial frequencies to be detected. This output frame is then run-length encoded as described above. The flowchart in figure 12 and C language code segment in figure 13 implements the ABC algorithm for one scan.

3.2 Comparison of Algorithms

The aim of the preprocessing algorithm is to accurately find the target within each frame. The rate at which this can be accomplished determines the maximum frame rate of the system. Computer simulations of the ability of

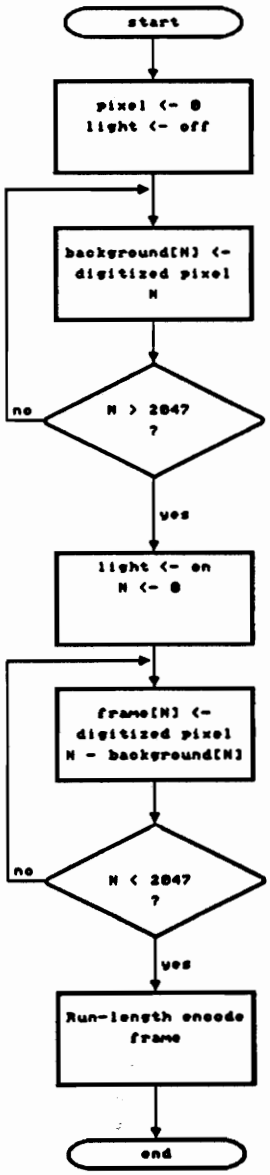


Figure 12. Flowchart of the ABC algorithm.

```
void ABC(int frame,int polarity, int transitions)
{
   int light, N,background[2048];

   light = 0;
   for(N = 0;N<2048;N++)
     background[N] = frame[N];
   light = 1;
   for(N = 0;N<2048;N++)
     frame[N] = frame[N] - background[N];
   run-length encode;
}
```

Figure 13. C language code for the ABC function.

each algorithm to find the target in a single representative frame of data were performed.

These simulations are used to merely benchmark the effectiveness of the algorithms in finding a target in a scene containing white noise. We assume that the target illuminates a single pixel with enough incident light that the output signal is always greater than the noise.

To determine a rigorous definition of the performance of each algorithm, one needs to derive models for the noise and targets. These models are then used as inputs to an exhaustive simulation of the performance of each algorithm. Plots of the simulated performance of each algorithm for different target-light levels (which are expressed as some multiple of the standard deviation of the noise) accurately indicate the best algorithm. This analysis, however, is beyond the scope of this thesis.

In this thesis, we simply evaluate the performance of each algorithm under the assumption that in laboratory environments we can always supply a target with a light level greater than the noise level.

The difference between the position generated by each algorithm and the actual position of the target in the analog signal is the measure used to compare the accuracies of the algorithms. Using figure 7 as a representative input frame, we simulated the three algorithms and determined the difference between the known and the measured positions. The known position was

found by giving the target a displacement with a micro-positioner and calculating the pixel position using the resolution relation in Chapter 2. The results of this simulation are summarized in Table 1.

The RLE and ABC algorithms determine the target location by averaging the two transition addresses associated with the target. However, since the actual position of the target could be anywhere between the two transition locations, the average is only an estimate of the target's position. As a result, the accuracy of the two algorithms is highly dependent on the threshold level. At lower thresholds, the error between the actual position and the average of the two transition locations may be large. On the other hand, the RM algorithm is not affected by this problem as it does not estimate the target's position. Because only one address is returned, there is no ambiguity in determining the target's location. However, the RM algorithm assumes that the target's location is, in fact, the brightest pixel.

As described earlier, the voltage level of the spike representing the target in a frame is proportional to the product of the light intensity and the exposure time. The exposure time is the reciprocal of the scan rate. As a result, as the scan rate increases the output voltage level corresponding to the target energy decreases. Consequently, as the voltage level of the target approaches that of the background, the user may encounter difficulty in setting the threshold in the RLE algorithm. If the threshold is set too low, the noise from either the background clutter or the sensor may register many false targets

Table 1. Comparison of the Performances of the Algorithms

Actual Position = 948

Algorithm	Calculated Position	Difference
RLE 50% threshold	972	24
RLE 60% threshold	964	16
RLE 70% threshold	951	3
RLE 80% threshold	955	7
RLE 90% threshold	947	1
RM	948	0
ABC 50% threshold	972	24
ABC 60% threshold	964	16
ABC 70% threshold	951	3
ABC 80% threshold	955	7
ABC 90% threshold	947	1

[19]. On the other hand, if the threshold is set too high, no targets will be found. Furthermore, because the threshold is not adaptive in the RLE algorithm, it cannot handle changes in target intensity between frames. The image intensity for a simple lens is proportional to the fourth power of cosine the angle relative to the optical axis [6]. Hence, the intensity of the target at the extreme ends of its path of motion (detected by the edges of the array) will be significantly lower than the intensity near the middle of its path (detected by the middle of the array.) If the threshold is set for a target voltage level corresponding to the center of the array, the algorithm will fail if the intensity of the target drops below the threshold at the extreme ends. If the threshold is set very low to account for a drop in the target intensity at the edges of the frame, the accuracy of the algorithm near center of the frame will be severely affected. The difficulty in setting the threshold is reduced by using the ABC algorithm because the background is subtracted from the frame. However, the ABC algorithm still suffers from the ambiguity as to where the target resides between the two transition addresses. Furthermore, the signal enhancement obtained by the background elimination assumes that there are no moving noise spikes. The RM algorithm is not affected by changing target voltage levels. However, it may fail if the target is not the brightest pixel [20].

The speed of each algorithm is taken to be the number of frames which can be received and processed per second. The RLE and RM algorithms are both implemented inline with the incoming data. Therefore, the number of frames processed equals the number of frames received. However, because of the

difficulty in setting the threshold in the RLE algorithm, the frame rate must be sufficiently low to allow the camera to provide a target voltage level significantly above the background level. The RM algorithm requires that the voltage level of the target be only relatively greater than the rest of the frame. Therefore, the frame rate can be much higher than that in the RLE algorithm. The ABC algorithm requires taking two frames for every processed frame. If the threshold can be set low enough to accommodate varying target voltage levels between frames, the maximum frame rate is one-half that in the case of RM algorithm.

3.3 *Implementation of Preprocessing Algorithms*

Because the output of the CCD array is sequential, the data from multiple independent sensors can be processed in parallel, but the data from individual sensors is most efficiently processed serially. Time constraints imposed by the algorithms limit the number of possible implementations. In each algorithm, a comparison of each pixel with either a threshold or a previous maximum must be performed. At the maximum pixel data-rate clock, 20 MHz, this comparison must be performed in less than 50 nanoseconds. In the ABC algorithm, a subtraction must be performed in this time period as well. The instruction cycle time of a standard microprocessor, however, is typically several hundred nanoseconds, which makes programmable implementations

of these algorithms difficult. Bit-slice processors, whose instruction execution cycles are typically around 150 nanoseconds, also suffer from this time constraint. In order to use either of these devices, one needs to implement a buffering scheme for storing the incoming data at the expense of increased hardware complexity and reduced throughput. Microcoded implementations require the sequencer to execute a sequence of microcode instructions in one period of the pixel clock. Since the algorithms require more than one instruction to be executed, these implementations require a sequencer clock frequency of over 40 MHz, which is impractical given the current technology.

Systolic implementations of these algorithms suffer in two ways. The size of the full systolic may be very large, and the serial nature of the input data limits the systolic's effectiveness. The latter limitation is due to what Kung [21] classifies as being "I/O bound." Systems which must perform only one computation per I/O request cannot achieve an increase in the execution speed using a systolic array because the bottleneck is in acquiring rather than processing the data [21]. The algorithms described in this chapter fall into the I/O bound category because one comparison must be made each time a new pixel arrives. In light of the time constraints placed on microprocessors and bit-slice processors and the fact that regular systolic architectures offer no decrease in I/O bandwidth, a hardwired implementation of the preprocessor algorithm is recommended. The following sections describe possible hardwired implementations for each algorithm. Although each can be

implemented using analog or digital techniques, the operations to be performed determine the most efficient method.

3.3.1 Run-Length Encoding Implementation

The RLE algorithm can be implemented easily using a single analog comparator with a slew rate fast enough to allow it to change at the clock rate. As each pixel is received, an address counter is incremented. This counter holds the address that corresponds to the spatial location of the pixel on the CCD array. If the output of the comparator is reset to zero at the beginning of each frame, the first pixel that is greater than the threshold will cause a transition from low to high on the comparator output. This transition is used to latch the value on the address counter into a first-in-first-out (FIFO) register. The output of the comparator will stay high until a pixel arrives which is less than the threshold. This pixel causes a transition from high to low on the comparator output, which is again used to latch the address on the counter into the FIFO. The FIFO can then be read by a microprocessor which determines the target's location from the addresses read from the FIFO. A digital implementation of the RLE digitizes the incoming signal and compares its values with a threshold value stored in a register or dip-switches. The addresses of transitions above and below the threshold value are then saved in a FIFO. In either implementation, if more than two addresses are generated by the RLE algorithm, the target's position must be predicted based on

previous frames using some form of estimation and tracking algorithm. Implementing any type of tracking algorithm would either dramatically increase the hardware complexity of the system, because systolic or wavefront architectures [21] are needed for providing real-time performance, or increase the software complexity, thereby eliminating real-time performance.

3.3.2 Implementation of Automatic Background Compensation

Storage of the background in the ABC algorithm requires the first scan in the process to be stored in a memory or FIFO. As the second scan is clocked out of the camera, the first scan must be subtracted from it. If the memory used to store the background frame is analog, the subtraction can be done in analog and the resulting frame run-length encoded as mentioned earlier. If the memory used is digital, the subtraction can be done with either discrete digital hardware or in software on a microprocessor. The resulting frame can then be run-length encoded using the digital implementation described above.

The digital implementation is the most practical due to the cost of analog memories. However, this implementation requires the RLE to be implemented digitally, which, as mentioned before, is more complicated.

3.3.3 Implementation of Running Maximum Algorithm

The RM algorithm requires the current pixel to be compared with the previous maximum, which is initialized to the lowest possible pixel value at the beginning of each frame. If an analog memory is used, the comparison can be done with an analog comparator. In either the analog or digital implementation, the address of the current maximum and its analog value must be saved after every comparison. The address of the current maximum is generated by a digital counter incremented at the clock rate. If digital memory is used to store the current maximum, the signal must be digitized and the comparison is done digitally. Again, the current maximum value and its corresponding address are saved.

3.4 Algorithm Selection

The run-length encoding algorithm, while easy to implement, does not provide high-frame rates because of the difficulty in setting the threshold. Further, the RLE cannot adapt to changing background conditions. The automatic background compensation algorithm easily adapts to changing target intensities. However, the extra overhead involved in storing the background frame seriously reduces its speed and increases its complexity. Based on the comparisons made in this Chapter, the running maximum provides the

greatest frame rate and accuracy. The ability of the RM algorithm to ignore changing target intensities allows it to perform well under varying background conditions. The algorithm may fail if the background clutter is brighter than the target. However, increasing the target's intensity at the source alleviates this problem without affecting the algorithm's accuracy. The relative simplicity in implementing the running maximum algorithm further adds to its strengths. The running maximum algorithm, therefore, is the algorithm that is used in this thesis to preprocess the pixel data.

4.0 Control and Processing System

4.1 Hardware

The custom control and processing System (CPS) is the heart of the measurement system. The CPS implements the running maximum preprocessing algorithm described in Chapter 3. The CPS also provides a frame grabber for acquisition of frames and an interface with a personal computer (PC) for post-processing, storage, and display of measurement data. The CPS is configured to control three cameras, allowing the system to acquire the directional data for three-dimensional motion measurements. By reducing the 2048 byte/frame, pixel stream generated by the CCD cameras to 2 byte/frame address streams, the CPS reduces the data bandwidth by more than three orders of magnitude. A behavioral model of the system was simulated using the VHDL hardware programming language [22] under the VM operating system on a Vax 11/785. The VHDL model was written and

simulated to provide qualitative information as to how the system should be interconnected and where possible bottlenecks occur. The VHDL source code is included in Appendix A and the schematics of the system are included in Appendix B.

The CPS hardware is comprised of four functional blocks as shown in figure 14. These blocks are: the system controller (SC), the digitizing unit, the running maximum unit, and the frame grabber. The individual blocks are described in the following sections. Each block represents a circuit board in the actual system. All boards were assembled on standard STD-BUS size cards, connected by a passive backplane, and enclosed in an STD-BUS rack. The circuits comprising the CPS hardware were simulated using the HILO3 hardware programming language under the HP/UX operating system on an HP 9000 minicomputer. The HILO3 model was written as a structural model [22] to determine quantitatively the exact timing of each circuit.

4.1.1 System Controller Board

The system controller (SC) consists of an Intel 8031 microcontroller with a clock speed of 10 MHz. The SC performs four major functions: an interface to the PC, timing generation, data acquisition, and data transfer. The firmware for the SC is stored in a 2Kx8 EEPROM. A block diagram of the SC board is shown in figure 15.

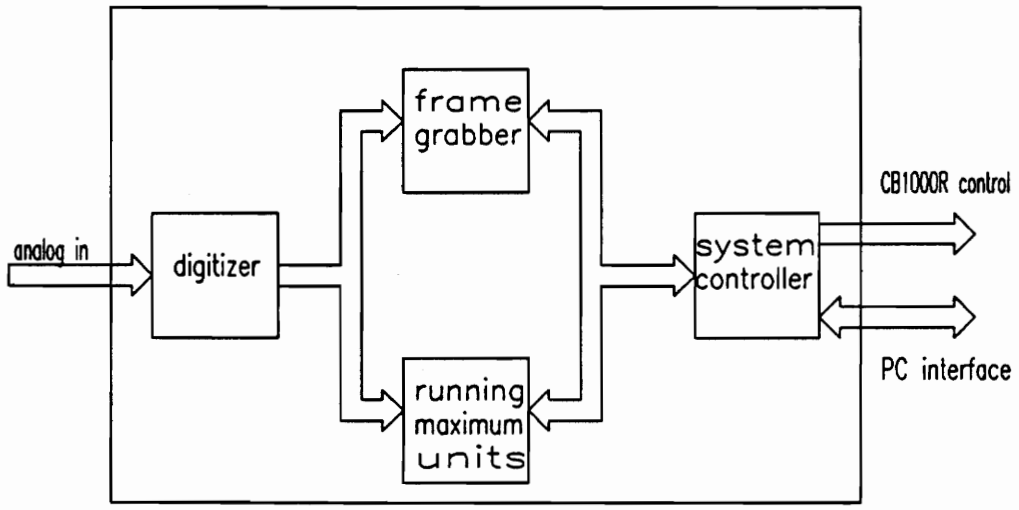


Figure 14. Block diagram of the CPS

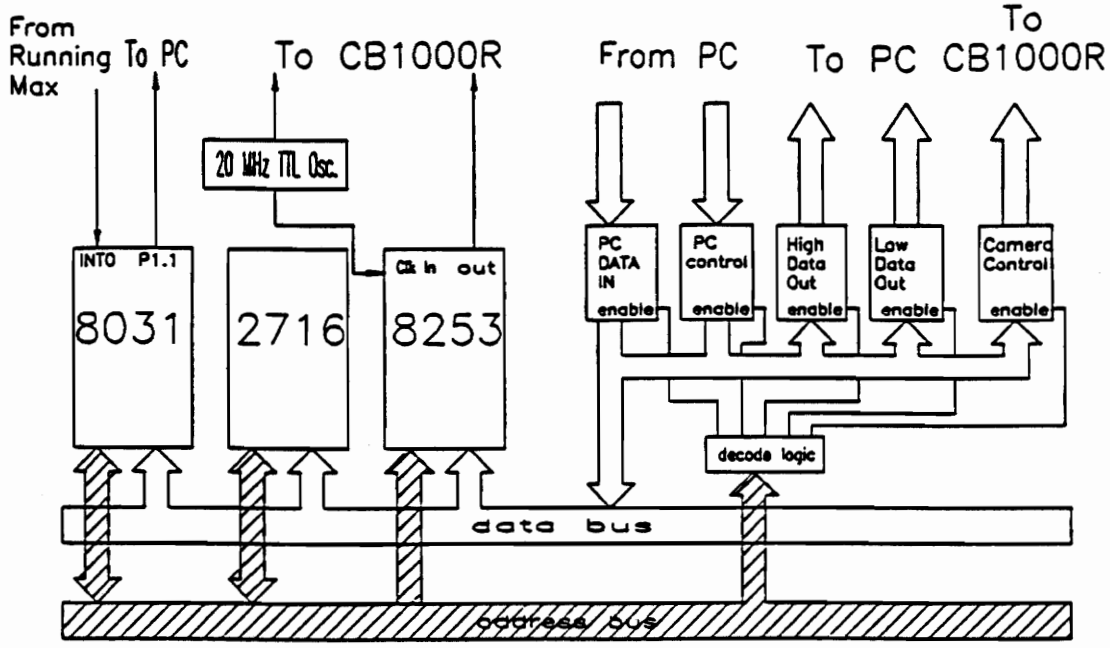


Figure 15. Block diagram of the SC

The SC maintains a bidirectional interface to the PC. This allows configuration data to be downloaded to the CPS and the acquired data from the CPS to be transferred to the PC. The interface protocol between the PC and CPS incorporates full-handshaking, allowing all transfers to take place asynchronously. In the configuration mode, the CPS receives data on its parallel data bus and clocks this data in when a DATA AVAILABLE strobe from the PC is received. The CPS returns an acknowledgement to the PC after the data has been read. In the data transfer mode, the CPS interrupts the PC when data is available, transmits the data, and waits for an acknowledgement from the PC before sending more data.

Timing generation for the pixel data-rate clock is accomplished by a 20 MHz TTL oscillator on the SC board. This clock is divided down to 2.5 MHz and is input to one timer in an Intel 8253 Programmable Timer. The 8253 divides the 2.5 MHz clock by a divisor provided by the PC to produce the frame-rate clock for the CCD cameras. This configuration allows the scan rate to be user programmable from the PC.

Two modes of data acquisition exist in the CPS. In the first mode, the SC simply reads the target addresses generated by the running maximum units. This is accomplished by servicing the interrupts generated by the running maximum units at the camera scan rate. In the second mode, the SC relinquishes its data and address busses to the frame grabber whose memory is then filled directly with the digitized pixel data. Once a frame has been

buffered, one of two data transfer functions occurs. The SC can either process the data locally and return only a target address or it can transfer the entire frame to the PC for post-processing.

4.1.2 Digitizing Unit

The digitizer for each of the three cameras consists of an Analog Devices AD9012 analog-to-digital (A/D) converter and an Analog Devices AD846 pre-amplifier. The AD9012 is an 8-bit flash converter capable of performing conversions at the 20 MHz pixel data-rate clock. The wide input bandwidth of the AD9012, 180 MHz, allows very accurate acquisition of high-speed pulse inputs without an external track-and-hold. The AD9012 requires an input voltage range of 0 to -2 volts. This requirement is met by inverting and amplifying the 0 to 1 volt camera signal with the AD846. The 8-bit converter provides 256 quantization levels of 7.8 mV per quantum. The reference voltage, -2.0 volts, needed by the AD9012 is provided by an Analog Devices AD707.

The high sample rate of the AD9012 is accomplished by eliminating the bus interface and end-of-conversion signal found on most A/D converters. The conversion begins on the rising edge of the encode clock and the result of the conversion becomes valid 13 ns after the next rising edge as shown in figure 16. Thus, the pixel stream is digitized by using the pixel data-rate clock and reading the data generated by the A/D prior to the next conversion.

Because the pixel intensity is digitized so that the position of the pixel with the greatest intensity can be found, the A/D needs only to be relatively accurate. By this we mean the actual value which the A/D gives a certain pixel is irrelevant provided its relative magnitude with respect to the rest of the pixels in the frame is correct.

4.1.3 Frame Grabber

The frame grabber enables the CPS to acquire entire data frames from one camera for storage and subsequent post-processing. This task is accomplished by completing a direct memory access (DMA) into the memory of the system controller. When a START-OF-FRAME pulse occurs, a binary counter on the frame grabber begins to increment at the pixel data-rate clock. The output of the binary counter is used as a pointer into the system controller's memory. When the one frame of 2048 bytes has been stored, a DONE flag is set. The DONE flag signals the SC that a frame has been acquired. Bus contention with the system controller is avoided by attaching all devices to the address and data busses with high impedance, TRI-STATE buffers. Once the pixel data is in memory, the system controller can either process the data or send the entire frame to the PC for display, storage, or post-processing.

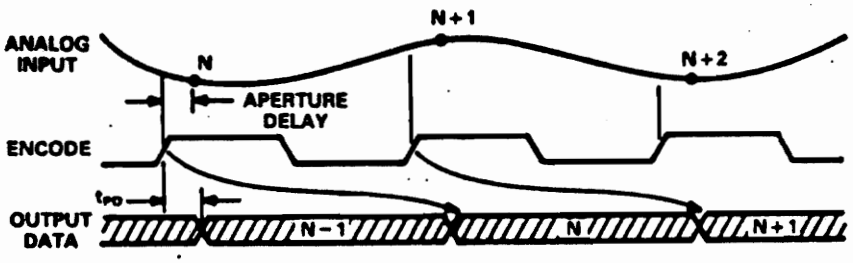


Figure 16. Timing Diagram for the A/D converter

4.1.4 Running Maximum Units

The CPS is configured to process the pixel data generated by three CCD cameras. Three running maximum units are implemented: a master unit and two slaves. Each running maximum unit implements the RM algorithm described in Chapter 3, however, because the A/D converter requires an input voltage range of 0 to -2 volts, the pixel with the highest intensity in each frame is given the lowest digital encoding. As a result, the running maximum units actually search for the minimum pixel in each frame. A block diagram of a running maximum unit is shown in figure 17.

A running maximum unit operates as follows. On the falling edge of the VALID VIDEO signal, the current minimum register is loaded with the hexadecimal value 0FF. This value is the maximum value the A/D converter can produce. The unit's 11-bit address generator is initialized to zero on this transition as well. When VALID VIDEO becomes true at the start of the next frame, the address generator is enabled and begins to increment in binary at the pixel data-rate clock. The digitized pixel is then compared to the current minimum register one per clock period. The comparison is performed by a 74LS682 8-bit comparator. The new pixel is the Q input and the current minimum is the P input. If the new pixel value is less than the current minimum, the $P > Q$ output of the comparator makes a high-to-low transition. This transition is used to load the current minimum register with the new pixel value. The transition is also used to load the minimum address register with the 11-bit value at the

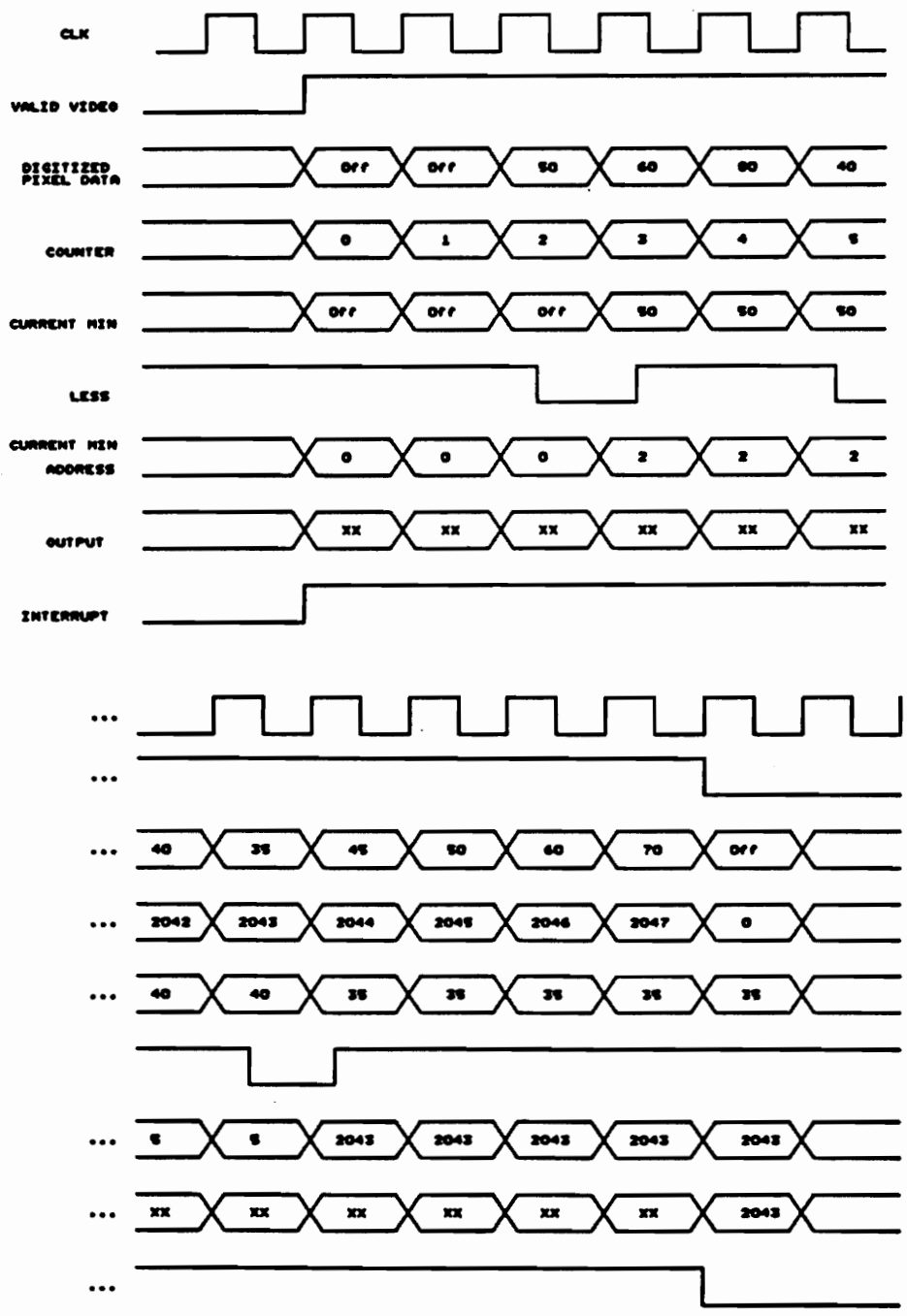


Figure 17. Block diagram of the Running Maximum unit.

output of the address generator. If the new pixel is greater than the current minimum, no action is taken and the running maximum unit waits for the next pixel. This process takes place once every clock period of the pixel data-rate clock. When 2048 clock periods have occurred, the VALID VIDEO signal makes a high-to-low transition. This transition latches the value of the minimum address register into the output register. This transition also instructs the master running maximum unit to interrupt the SC. The running maximum unit is then ready to search the next frame of incoming pixels. A timing diagram of the operation of the master running maximum unit is given in figure 18.

Once the master running maximum unit has interrupted the SC, the SC has one frame period to read the target address values stored in the output registers of the master unit and the two slaves. The slave running maximum units perform the search operation in parallel with the master unit, however, they do not generate an interrupt at the end of a frame. Because the cameras are operating on the same pixel data-rate clock, their VALID VIDEO signals are identical. As a result, the target address generated by the master running maximum unit becomes valid within one clock period of those generated by the other two slaves.

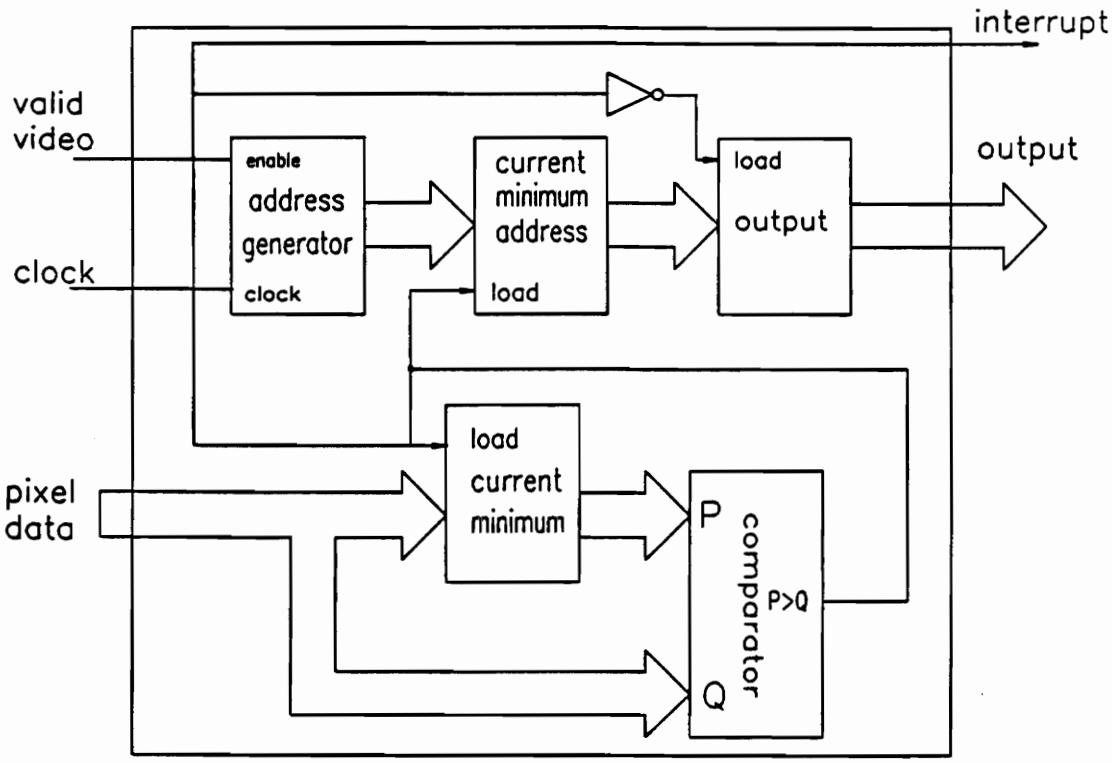


Figure 18. Master Running Maximum unit timing diagram.

4.2 Firmware

As mentioned earlier, the firmware which programs the microcontroller resides in a 2Kx8 EEPROM on the SC board. A flowchart of the firmware is shown in figures 19 through 22.

After a hardware reset, the firmware waits for a DATA AVAILABLE signal from the PC by polling the DATA AVAILABLE input bit P1.0 on the 8031. Once this has been received, the high byte of the divisor, to be programmed into the 8253 programmable timer, is read. The firmware returns an ACKNOWLEDGE signal to the PC and then waits for another DATA AVAILABLE. Following the receipt of the next DATA AVAILABLE signal, the low byte of the divisor is read and an ACKNOWLEDGE is returned to the PC. The handshaking protocol is performed once more to receive a mode byte from the PC. Once the timer divisor and mode bytes have been received, the firmware programs the 8253 timer to operate in its rate generator mode.

The firmware branches to one of three routines based on the value of the mode byte received from the PC. If the mode byte is 'zero' the firmware prepares to service the interrupts from the running maximum units. The corresponding interrupt service routine reads the three output ports of the three units and transfers the data to the PC employing the full handshaking protocol. If the mode byte is 'one' or 'two', the firmware relinquishes its address and data busses to the frame grabber and polls the DONE FLAG generated by the frame

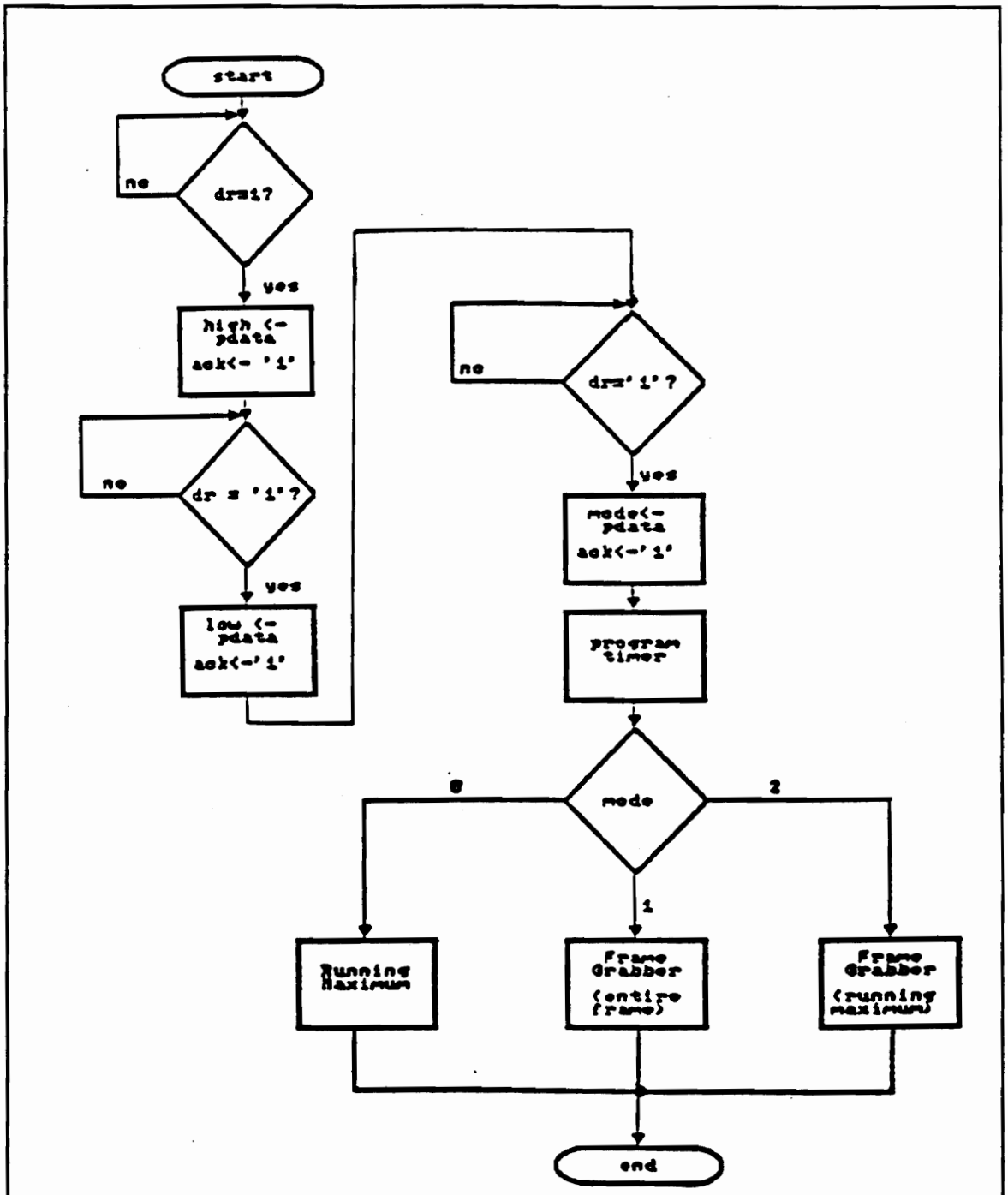


Figure 19. Flowchart for the 8031 microcontroller firmware

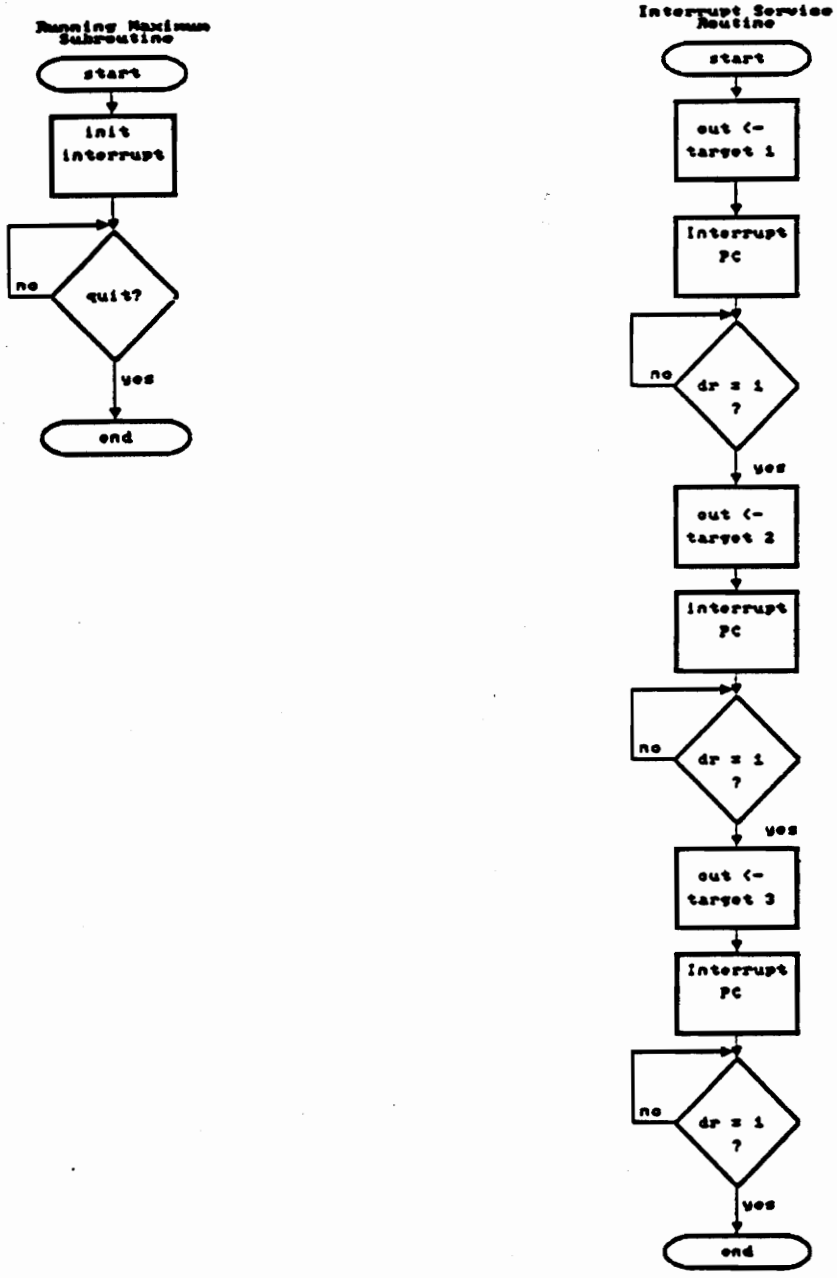


Figure 20. Flowchart for the running maximum subroutine

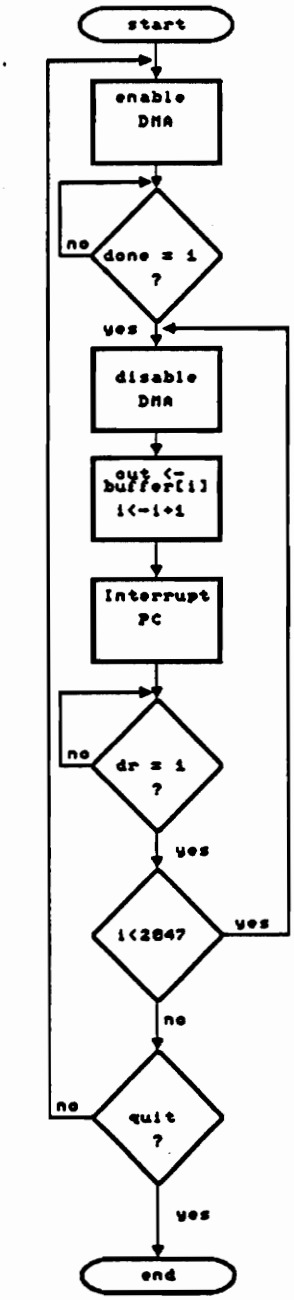


Figure 21. Flowchart for the frame grabber entire frame subroutine

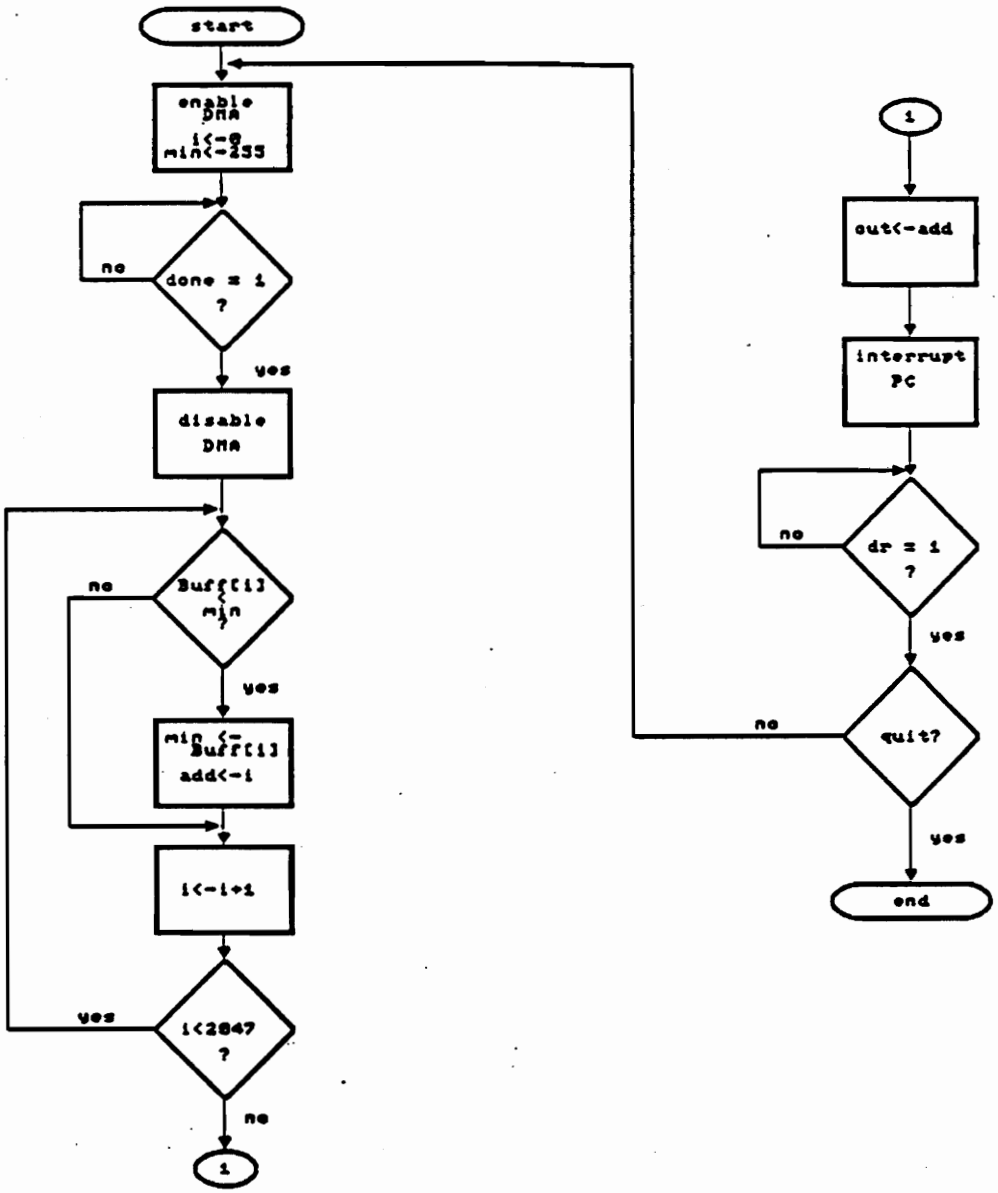


Figure 22. Flowchart for the frame grabber running maximum subroutine

grabber. In mode 'one', the SC transfers the entire frame of data to the PC. In mode 2, the SC searches the data for the minimum value and transfers its address to the PC. This algorithm can be replaced with another target locating algorithm if desired.

In either mode 'one' or mode 'two', the frame rate must be slow enough for the microcontroller to access each pixel stored in memory. As the minimum access time of the 8031 is 4 microseconds including the address setup time, the minimum time between frames must be

$$2048 \text{ pixels} \times \frac{4 \text{ microsecond}}{\text{pixel}} = 8.2 \text{ milliseconds} \tag{11}$$

Thus, the maximum frame rate is

$$\frac{1}{8.2 \text{ milliseconds}} = 122 \text{ Hz} \tag{12}$$

or 122 frames per second.

In mode 'zero', the running maximum units perform the linear search operation as the data is shifted out of the camera and digitized and can, therefore, processes the data at the frame rate. The process takes 2048 cycles of the pixel data-rate clock and is not a function of the frame-rate clock.

4.3 Software

The software on the PC performs three major functions: data acquisition, storage, and display. A flowchart of the software running on the PC is shown in figure 23. The operation of the software is as follows. The software issues a reset signal which initializes the program counter in the SC's microcontroller to zero. The microcontroller then waits for configuration data. The software calculates the high and low bytes of the timer divisor from the decimal value entered by the operator and downloads them to the CPS over the parallel link. The software also sends a mode byte to program the CPS to perform the desired data acquisition and processing function.

Once the CPS has been programmed, the PC software prepares to receive data from the CPS. This involves setting up the interrupt service routine and configuring the buffers to either accept 8-bit frame data or 11-bit target address data. A graphical user interface plots the received data in one of three formats. If frame data is being received, the software waits for an entire frame to be received and then plots the digitized frame in the form of a voltage versus pixel value. If target addresses are being received, the software either plots the target address versus time or plots the target address from one camera against the target address from another. In either case, the computer graphic display is updated with every new address or pair of addresses. The computer display can update the displacement or relationship between two

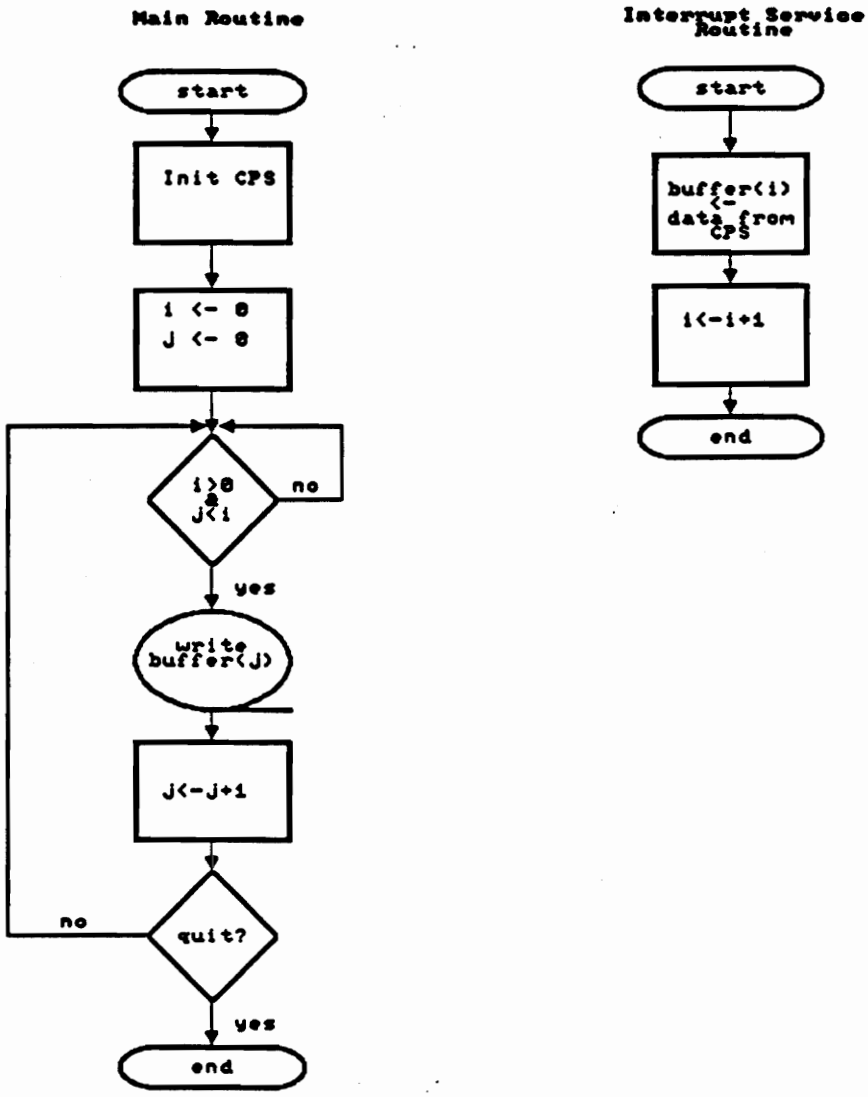


Figure 23. Flowchart for PC software

cameras at frame rates up to 1500 Hz. If the frame rate is faster, the display lags behind the motion slightly.

The most important task of the software is to store the data on the hard drive, as it allows the data from experiments to be saved for further processing. The storage of the target address data is performed in two steps. The data is first buffered in RAM and then transferred onto the hard drive. Pointers into the buffer allow the interrupt service routine to fill the buffer at a different rate than the hard drive can empty it. The data is written to the hard drive in binary format to reduce the space needed for the data. The number of addresses to be saved is specified by the operator.

5.0 Results

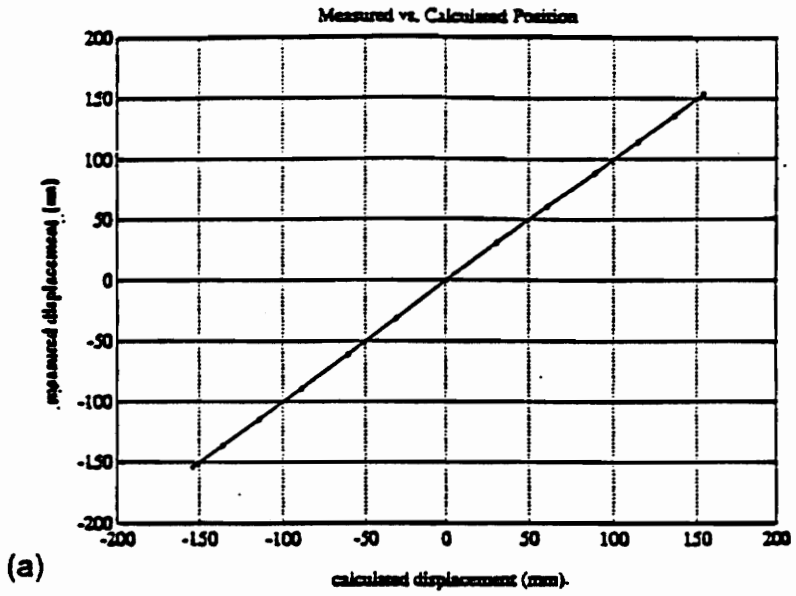
The results of the VHDL and HILO3 simulations indicate that the CPS can process the data from three cameras in parallel at a rate of 20 Mbytes/sec for each camera or 60 Mbytes/sec total. The system can therefore process the data generated by the CCD1500R cameras when they are run at their maximum frame rate. After the HILO3 simulations quantitatively verified the operation of the design, the system was constructed.

Three tests were performed to verify the system's operation. The first was a static test to determine its linearity. A 1 gram, micro incandescent bulb was mounted to a precision pendulum. The resolution of each pixel was determined by finding the displacement of the pendulum which registered full scale on the array. This displacement was then divided by 2048 to determine the pixel resolution. For the configuration tested, a 327.8 mm displacement of the pendulum yielded a pixel resolution of 160 microns. The pendulum was given a known angular displacement using a precision micrometer. The

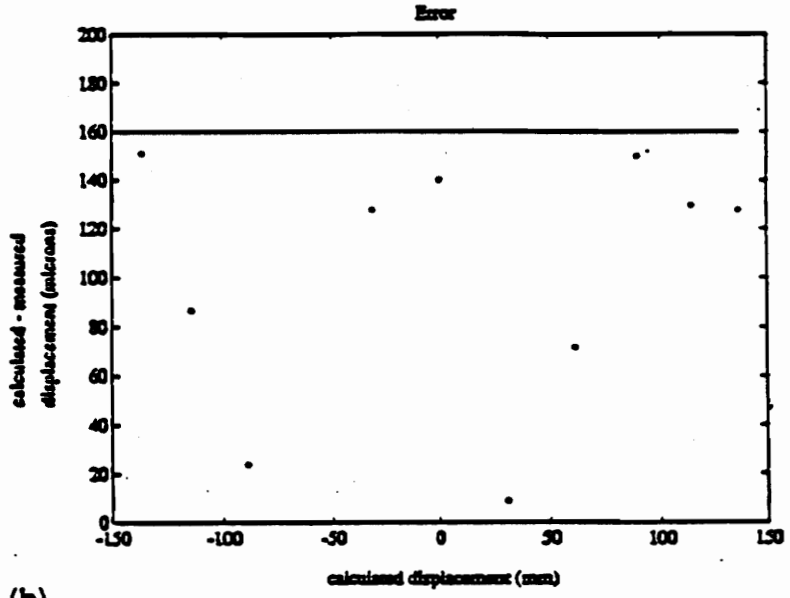
angular displacement was then converted to a linear horizontal displacement. The measured horizontal displacements are shown in figure 24(a). The differences between the measured and calculated displacements were then plotted to verify the resolution relation given in Chapter 2. This plot is shown in figure 24(b). The error between the expected and measured values is always less than 160 microns.

The pendulum was used in the static calibration to test the ability of the cylindrical lens to compress the scene in front of it to a line image. The results of the static calibration indicate that the circular motion of the pendulum had no negative effect on the accuracy or linearity of the system.

The second test performed was a dynamic measurement comparison between a strain gage and the optical measurement system. This gives a benchmark to which the optical system can be compared. The motion of a two-mass, two-beam system was measured using both a strain gage and our optical measurement system. The structure was excited at a frequency at which the response of the structure could be accurately measured with a strain gage. Again, a micro incandescent bulb provided illumination for the optical system. The frequency content of their outputs are compared in figures 25(a) and 25(b). The excellent agreement between the frequency spectrum obtained with the strain gauge and the frequency spectrum obtained with the optical system indicates that the optical system performs as well as the strain-gage system when no out-of-plane motion exists.



(a)



(b)

Figure 24. Calibration curve and resolution scatter plot: (a) strain gauge (b) optical system

Testing the ability of the optical system to measure complex, out-of-plane motions is very difficult because there are no results to compare with. However, the fact that the optical measurement system is not attached to the structure and the three planes of motion are measured independently indicate that out-of-plane motions should have no effect on the accuracy of the system.

To illustrate the ability of the optical system to measure complex, out-of-plane motions, a reflector was attached to the tip of a circular rod. The rod was chosen because it cannot easily be instrumented with other measurement systems. The rod was parametrically excited such that the expected response was quasi-periodic motion [23]. The motion of the tip of the rod is plotted vs. time and as a pseudo-phase plane in figures 26(a) and 26(b). The frequencies of the large loop and the smaller outer loops in the phase plane correspond to the responses of the first and third modes, respectively. The closed Poincare' curve, shown in figure 27, indicates that these two frequencies are not commensurable and therefore, the motion is quasi-periodic as expected. The qualitative agreement between the theory and measured experimental data indicates that the optical measurement system can measure complex, out-of-plane motions correctly.

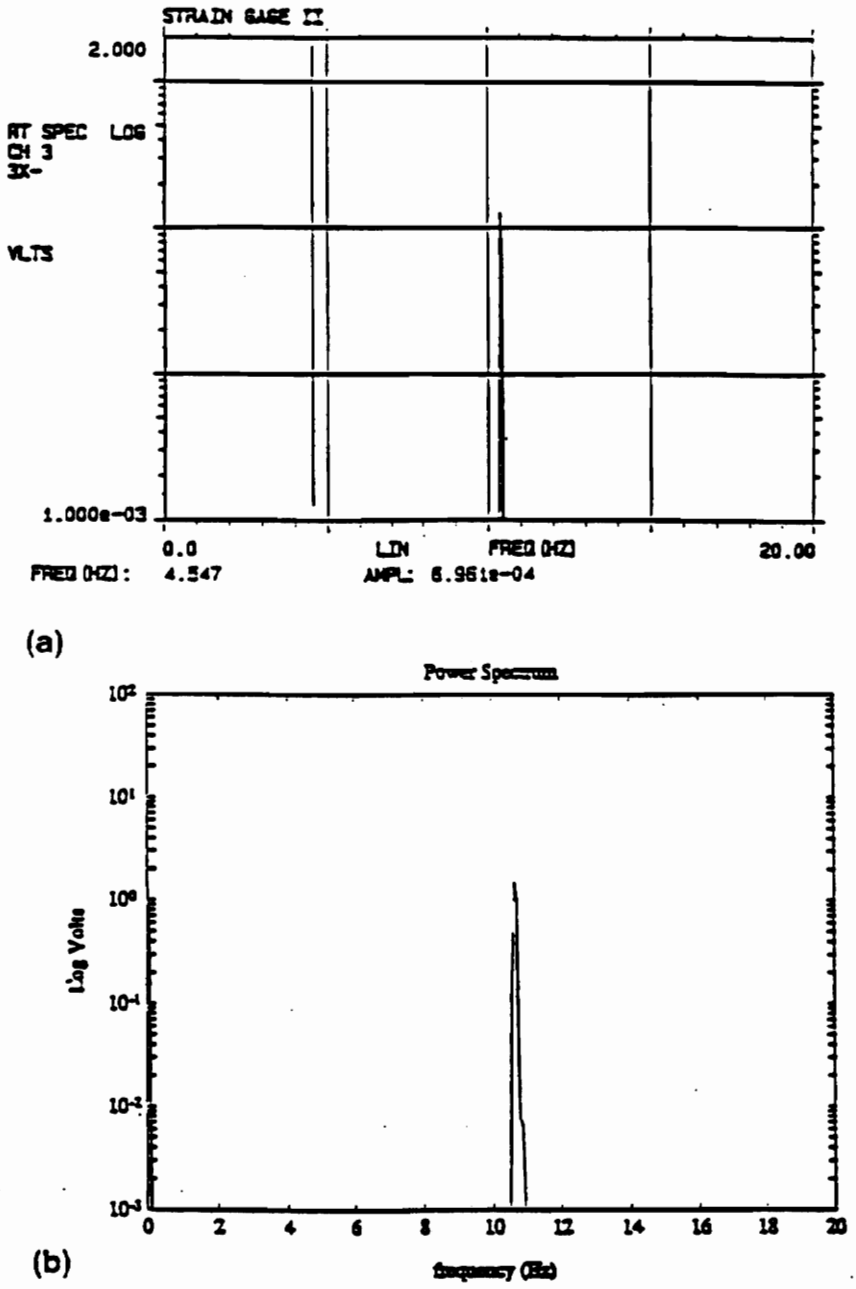


Figure 25. Comparison of optical system output and strain gauge output.: (a) strain gauge (b) optical system

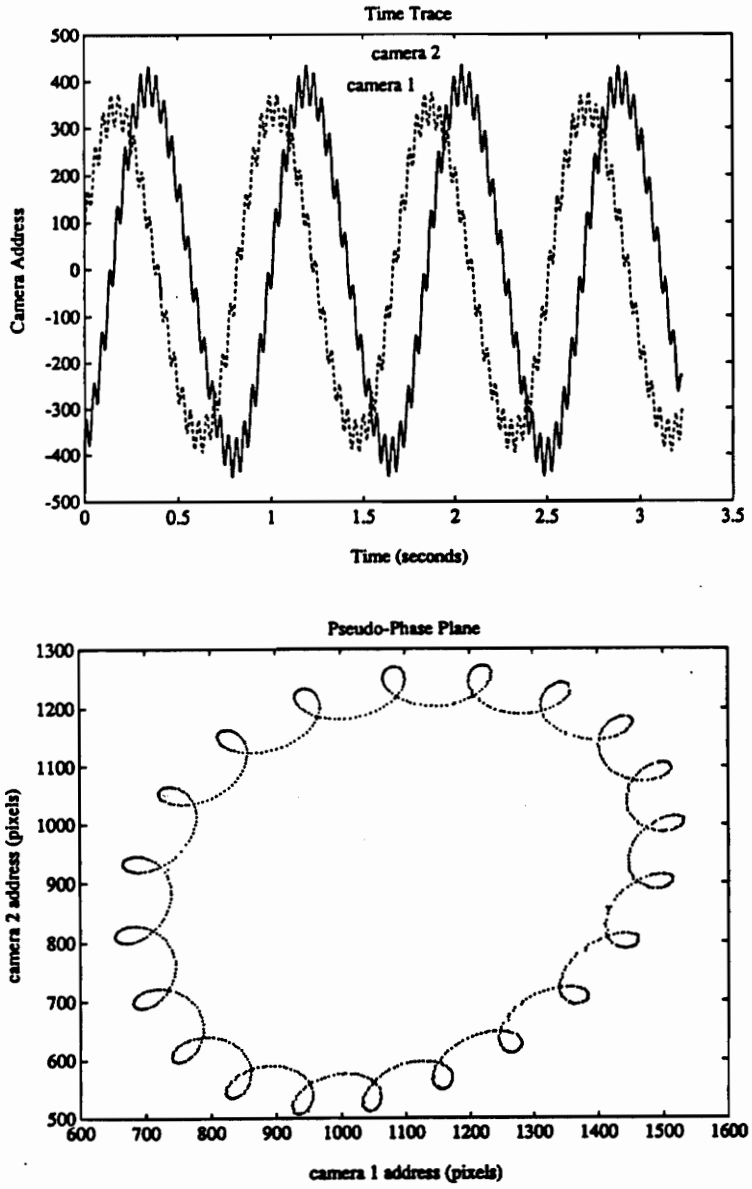


Figure 26. **Motion of a circular rod under parametric excitation.:** (a) time trace (b) pseudo-phase plane

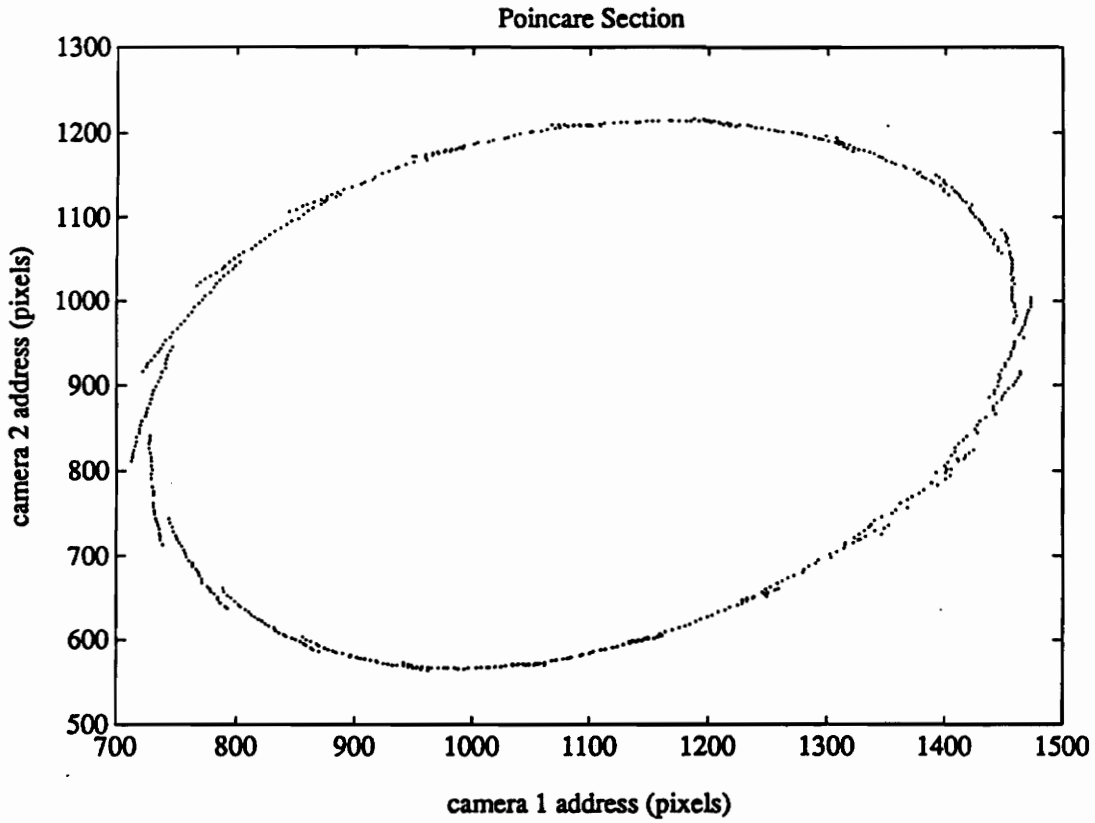


Figure 27. Poincare' section

6.0 Conclusions and Recommendations for Further Research

Performing the preprocessing tasks, described by Fathi and Fines [18] and Pfister [19], on the high-rate and bandwidth data streams generated by the optical measurement system is difficult due to time limitations. Despite its simplicity, the algorithm implemented for performing the preprocessing task effectively reduces the data bandwidth to a rate manageable by the host PC. As the results indicate, the Seymour FDMS optical measurement system coupled with the custom control and processing system described in this thesis is capable of providing accurate measurements of the motion of lightweight flexible structures. The custom CPS electronics eliminate the need for expensive and powerful computing systems without sacrificing performance.

6.1 Selection of Measurement System

Based on the strengths of the Seymour FDMS, the system was chosen as a basis for an improved optical measurement system. The ability of preprocessing hardware to reduce the data bandwidth of optical measurement systems prompted an investigation into several preprocessing algorithms and architectures. A method for making measurements with a one-axis sensor was developed. In addition, a method to fuse the data from three cameras to yield three-dimensional displacement measurements was described.

6.2 Algorithm Development and Evaluation

Three preprocessing algorithms were introduced in Section 3.1. These algorithms were chosen for evaluation based on previous work done on systems providing similar data. The first, run-length encoding, has serious limitations caused by its fixed threshold. The second, running maximum, allows accurate detection of the signal in all instances except where the clutter is brighter than the target. The third algorithm, automatic background compensation, adds the overhead of background storage and suffers from some of the limitations of the RLE algorithm. Using raw frame data acquired with a digital frame grabber, we performed simulations of the ability of the algorithms to find the target in the frame and discussed them in Section 3.2.

A heuristic comparison of the relative speeds of the algorithms was also made in that section. In Section 3.3, the computing requirements of the preprocessing task and implementations of the three algorithms were discussed. Consequently, the running maximum algorithm was chosen based on its accuracy, speed, and ease of implementation.

6.3 *Algorithm Implementation*

The running maximum preprocessing algorithm was implemented using a hardwired approach. The hardware implementation, while not easily changed, provides excellent performance, allowing the cameras to be operated at their maximum frame rate. The design of the bi-directional interface allows the storage and display capabilities of a standard microcomputer to be utilized.

6.4 *Design Improvements*

The system described in this thesis is a prototype system. Therefore, there are several design improvements which can be made to enhance its performance. The first of these improvements is the use of an automatic gain control on the analog input. If the voltage level of the input to the A/D converter can be kept near the -2 volts, more of the range of the A/D could be used and the accuracy

of the determination of the greatest value will be improved. The system controller firmware can be modified to send only the difference in the displacement in each frame. This will further reduce the data bandwidth. However, if the reference is lost the data will be invalid. The addition of real-time FFT capabilities will make the system even better suited for performing real-time motion measurement and analysis.

References Cited

[1] P. Horowitz and W. Hill, **The Art of Electronics,** New York: Cambridge Press, 1980, pp. 601-602.

[2] B. Balachandran, "A Theoretical and Experimental Study of Modal Interactions in Resonantly Forced Structures," Ph.D., Virginia Polytechnic Institute and State University, Blacksburg, VA, 1990.

[3] M. Serridge and T. R. Licht, **Piezoelectric Accelerometer and Vibration Preamplifiers,** Denmark: Bruel & Kjaer, 1986, pp. 13-14.

[4] I. Fehervari, P. Lambrechts, E. Bactens, and A. Oosterlinck, "Multiprocessor for Vision," The Society of Photo-Optical Instrumentation Engineers, Vol. 804, Advances in Image Processing, pp. 84-91, 1987.

[5] S. Welch, "Optical Distributed Sensing and Computation For A Proposed Flexible Beam Experiment," Seventh VPI&SU/AIAA Symposium on Dynamics and Control of Large Structures, Blacksburg, VA, 1989.

[6] G. Uber, "Illumination and Imaging of Moving Objects," The Society of Photo-Optical Instrumentation Engineers, Vol. 1006, Optics, Illumination and Image Sensing for Machine Vision III, pp. 2-5, 1988.

[7] A. H. Bhuiya and A. Farhat, "Model of a Fast Vision Detector for Determination of the Velocity of a Moving Object," IEEE Transactions on Instrumentation and Measurement, Vol. IM-35, pp. 585-590, 1986.

[8] P. T. Kotnik, S. Yurkovich, and U. Ozguner, "Acceleration Feedback for Control of a Flexible Manipulator Arm," Journal of Robotic Systems, Vol. 5, pp. 181-196, 1988.

[9] U. Ozguner, S. T. Yurkovich, J. W. Martin, and P. T. Kotnik, "Laboratory Facility of Flexible Structure Control Experiments," IEEE Control Systems Magazine, Vol. 8, pp. 27-33, 1988.

[10] H. Seymour, "Electro-Optic Space Positioner," United States Patent #4,136,568, 1979.

[11] V. DeAngelis, "In-Flight Deflection Measurement of the HiMAT Aeroelastically Tailored Wing," Journal of Aircraft, Vol. 19, 1982.

[12] V. DeAngelis, "Electro-Optical Flight Detection Measurement System," Society of Flight Test Engineers, 18th Annual Symposium, pp. 22-1 to 22-14, 1987.

[13] I. Fehervari, P. Lambrechts, E. Bactens, and A. Oosterlinck, "Fast Measurement of Physical Dimensions," The Society of Photo-Optical Instrumentation Engineers, Vol. 804, Advances in Image Processing, pp. 278-283, 1987.

[14] J. Van de Hende and O. Constandse, "A System for High Speed Signal Processing: the Application of Tightly Coupled Processor Architectures," IEEE Symposium on Real-Time Systems, Proceedings, pp. 185-189, 1983.

[15] R. Bermback, H. Sharfenberg, and H. Gambel, "Industrial Image Processing System Allows Parallel Computing on Sensor Data," The Society of Photo-Optical Instrumentation Engineers, Vol. 804, Advances in Image Processing, pp. 269-277, 1987.

[16] N. Wilkinsin, M. Atkins, and J. Rogers, "A Real-Time Data Acquisition System," IEEE Symposium on Real-Time Systems, Proceedings, pp. 54-59, 1989.

[17] "CCD The Solid State Imaging Technology," Fairchild Weston, pp. 171-175, 1984.

[18] E. Fathi and F. Fines, "Real-Time Data Acquisition, Processing and Distribution for Radar Applications," IEEE Symposium on Real-Time Systems, Proceedings, pp. 95-104, 1984.

[19] H. Pfister, "Image Processing for a Combined Starring and Scanning IR Sensor," The Society of Photo-Optical Instrumentation Engineers, Vol. 804, Advances in Image Processing, pp. 225-228, 1987.

[20] H. Hansen and C. Elyashar, "Adaptive Threshold Adjustment and Control," The Society of Photo-Optical Instrumentation Engineers, Vol. 1096, Signal and Data Processing of Small Targets, pp. 44-54, 1989.

[21] H. T. Kung, "Why Systolic Architectures?," IEEE Computer, Vol. 18, pp. 37-45, 1982.

[22] J. R. Armstrong, **Chip-Level Modeling with VHDL,** Prentice Hall, N.J., 1989.

[23] P. F. Pai, "Nonlinear Flexural-Flexural-Torsional Dynamics of Metallic and Composite Beams," Ph.D., Virginia Polytechnic Institute and State University, Blacksburg, VA, May 1990.

Appendix A. VHDL Source Code

```
------------------------------------------------------------------------
-- Entity: TEST BENCH
-- Description:
-- The entities comprising the CPS are interconnected in the TEST
-- BENCH entity.  Furthermore, the input from the external world
-- (ie. keyboards, power) are provided by the TEST BENCH.
------------------------------------------------------------------------
entity TEST_BENCH is
end TEST_BENCH;

use work.all;
architecture CPS of TEST_BENCH is
signal p_clk,frame,vv,sclk,run,r_int,power,reset,dr,ack,p_int,ret : bit;
signal vv2,vv3,sclk2,sclk3 : bit;
signal cs,cs2,cs3: real;
signal camcon,pdata,ds1,ds2,ds3,keybd : bit_vector(7 downto 0);
signal target1,target2,target3,t_data: bit_vector(10 downto 0);

component camera1
  generic(num_frame: integer; name: string(1 to 8));

  port(p_clk,frame : in bit;
       camcon: in bit_vector(7 downto 0);
       vv : inout bit;
       sclk: out bit;
       cs: out real);
end component;

component digitizer
  port(cs : in real;
       sclk: in bit;
       ds: out bit_vector(7 downto 0));
end component;

component rmu_master
   port(vv,sclk : in bit;
        ds : in bit_vector(7 downto 0);
        target: out bit_vector(10 downto 0);
        r_int: inout bit);
end component;

component rmu_slave
```

```
  port(vv,sclk: in bit;
       ds : in bit_vector(7 downto 0);
       target: out bit_vector(10 downto 0);
       r_int : in bit);
end component;

component scu
  port(r_int,reset,power,dr: in bit;
       t_data: out bit_vector(10 downto 0);
       target1,target2,target3: in bit_vector(10 downto 0);
       pdata : in bit_vector(7 downto 0);
       camcon : out bit_vector(7 downto 0);
       p_int,ack,frame: out bit;
       p_clk : inout bit);
end component;

component pc
  port(p_int,ack,power,ret : in bit;
       t_data:  in bit_vector(10 downto 0);
       keybd: in bit_vector(7 downto 0);
       reset: out bit;
       pdata: out bit_vector(7 downto 0);
       dr : inout bit);
end component;

for all: camera1 use entity work.camera1(behavior);
for all: digitizer use entity work.digitizer(behavior);
for L3: rmu_master use entity work.rmu_master(behavior);
for L4: scu use entity work.scu(behavior);
for L5: pc use entity work.pc(behavior);
for all: rmu_slave use entity work.rmu_slave(behavior);

begin
-- System Interconnection
  L1: camera1
      generic map(10,"cam1.dat")
      port map(p_clk,frame,camcon,vv,sclk,cs);

  L2: digitizer
      port map(cs,sclk,ds1);

  L3: rmu_master
      port map(vv,sclk,ds1,target1,r_int);

  L4: scu
   port map(r_int,reset,power,dr,t_data,target1,target2,target3,pdata,camcon,
         p_int,ack,frame,p_clk);
  L5: pc
   port map(p_int,ack,power,ret,t_data,keybd,reset,pdata,dr);

  L6: camera1
    generic map(10,"cam2.dat")
    port map(p_clk,frame,camcon,vv2,sclk2,cs2);

  L7: camera1
     generic map(10,"cam3.dat")
```

```
    port map(p_clk,frame,camcon,vv3,sclk3,cs3);

  L8: rmu_slave
    port map(vv2,sclk2,ds2,target2,r_int);

  L9: rmu_slave
    port map(vv3,sclk3,ds3,target3,r_int);

  L10: digitizer
      port map(cs2,sclk2,ds2);

  L11: digitizer
      port map(cs3,sclk3,ds3);

  run <= '1';
  process(run)
  begin
    power <= transport '1' after 1 ns;
    keybd <= transport "00000000" after 10 ns;
    ret <= transport '1' after 20 ns;
    ret <= transport '0' after 22 ns;
    keybd <= transport "00000010" after 50 ns;
    ret <= transport '1' after 60 ns;
    ret <= transport '0' after 62 ns;
    keybd <= transport "00000001" after 100 ns;
    ret <= transport '1' after 110 ns;
    ret <= transport '0' after 112 ns;
    power <= transport '0' after 15000 ns;
  end process;
end CPS;
```

```
------------------------------------------------------------------
-- Entity: PC
-- Description:
-- The PC entity models the functions performed by a Personal computer.
-- The functions include the programming of the SCU, and the aquisition
-- and storage of the target addresses generated by the Running Maximum
-- units.
--
-- Input signals:
--              p_int: interrupt signal from the SCU
--              ack : acknowledge signal from the SCU
--              power: enable signal from the test bench.
--              ret : data ready signal from the test bench
--              t_data: data byte from SCU
--              keybd: data byte from the test bench
--
-- Ouput signals:
--              reset: reset signal to SCU
--              pdata: data byte to SCU
--              dr : data ready signal to SCU
-- Generics:
--
--
------------------------------------------------------------------
use work.cv.all;
entity pc is
  port(p_int,ack,power,ret: in bit;
       t_data : in bit_vector(10 downto 0) := "00000000000";
       keybd : in bit_vector(7 downto 0);
       reset : out bit;
       pdata : out bit_vector(7 downto 0);
       dr : inout bit);

end pc;

architecture behavior of pc is
type tadd_array is array(1 to 3) of tadd;
signal dr1,dr2,send : bit;
signal hi,lo,mode : bit_vector(7 downto 0);
file target : tfile is out "target.dat";
begin

------------------------------------------------------------------
-- Process :  SETUP
--
-- Description:
-- The SETUP process models the behavior keyboard input.
-- The process waits for the POWER signal to become true.  Upon
-- receipt of the RET signal, the keyboard is read.  This occurs three
-- times.  The first two values are the high and low bytes of the divisor
-- for the timer process in the SCU entity, and the third a mode bit.
--
-- Sensitivity List: empty.
--
-- Functions called: none.
------------------------------------------------------------------
```

```
 setup : process
        begin
          wait until power = '1' and power'event;
          wait until ret = '1';
          wait until ret = '0';
          hi <=keybd;
          wait until ret = '1';
          wait until ret = '0';
          lo <= keybd;
          wait until ret = '1';
          wait until ret = '0';
          mode <= keybd;
          send <= transport '1';
        end process setup;

-------------------------------------------------------------------------
-- Process :  DOWNLOAD
--
-- Description:
-- The DOWNLOAD process accepts the timer divisor from the PC
-- using a full handshake protocol.
--
-- Sensitivity List: empty.
--
-- Functions called: none.
-------------------------------------------------------------------------
  download: process
          begin
            wait until send'event;
            if send = '1' then
              pdata <= hi;
              reset <= '1';
              dr1 <= transport '1' after 1 ns;
              dr1 <= transport '0' after 10 ns;
              wait until ack = '1';
              wait until ack = '0';
              pdata <= lo;
              dr1 <= transport '1' after 1 ns;
              dr1 <= transport '0' after 10 ns;
              wait until ack = '1';
              wait until ack = '0';
              pdata <= mode;
              dr1 <= transport '1' after 1 ns;
              dr1 <= transport '0' after 10 ns;
              wait until ack = '1';
              wait until ack = '0';
            end if;
          end process download;


-------------------------------------------------------------------------
-- Process : DAS
--
-- Description:
-- Upon receipt of an interrupt from the SCU, the DAS process reads
-- the value on the t_data signal and writes it to the hard disk.
```

```
--
-- Sensitivity List: empty.
--
-- Functions called: none.
-------------------------------------------------------------------------
  DAS: process
       variable tgt : tadd_array;
       variable t: integer := 1;
       begin
         wait until p_int'event;
         if p_int = '1' then
           tgt(t) := t_to_st(t_data,t);
           dr2 <= transport '1' after 1 ns;
           dr2 <= transport '0' after 20 ns;
           t:=t + 1;
           if t = 4 then
             t := 1;
             for i in 1 to 3 loop
              write(target,tgt(i));
             end loop;
           end if;
         end if;
       end process DAS;
 -------------------------------------------------------------------------
   dr <= dr1 when not dr1'quiet else
         dr2 when not dr2'quiet else
         dr;

end behavior;
```

```
--------------------------------------------------------------------
-- Entity: SCU
-- Description:
-- This process models the functions performed by an Intel 8031
-- microcontroller, an Intel 8253 programmable timer, and a quartz
-- oscillator.
--
-- Input signals:
--              r_int: end of frame interrupt from SCU
--              reset: reset signal from PC
--              power: quartz oscillator enable
--              dr: data ready from PC
--              target1,target2,target3: target addresses from the
--                                  RMU_MASTER and 2 RMU_SLAVES.
--              p_data: data byte from PC
-- Ouput signals:
--              t_data: data byte to PC
--              camcon: camera control word
--              p_int: interrupt signal to PC
--              ack: acknowledge signal to PC
--              frame: frame rate clock
--              p_clk: pixel rate clock

-- Generics:  none.
--------------------------------------------------------------------
use work.cv.all;
entity scu is
  port(r_int,reset,power,dr: in bit;
       t_data : out bit_vector(10 downto 0);
       target1 : in bit_vector(10 downto 0) := "00000000000";
       target2 : in bit_vector(10 downto 0) := "00000000000";
       target3 : in bit_vector(10 downto 0) := "00000000000";
       pdata: in bit_vector(7 downto 0) := "00000000";
       camcon : out bit_vector(7 downto 0);
       p_int,ack,frame: out bit;
       p_clk : inout bit);
end scu;

architecture behavior of scu is
signal hi,lo,mode: bit_vector(7 downto 0);
signal divisor: integer;
signal gate,t_clk: bit;

begin
--------------------------------------------------------------------
-- Process : CRYSTAL
--
-- Description:
-- This process models a 20 MHz quartz oscillator.  The POWER signal
-- enables the CRYSTAL process
--
-- Sensitivity List: power
--
-- Functions called: none
--------------------------------------------------------------------
  crystal: process(power,p_clk)
```

```
        begin
         if power = '1' then
           p_clk <= transport not p_clk after 25 ns;
         end if;
        end process crystal;

-------------------------------------------------------------------------
-- Process : DIVIDE
--
-- Description:
-- This pprocess models the behavior of a 74LS93 binary counter.  The
-- process divides the 20 MHz quartz crystal frequency down to 2.5 MHz.
-- The 2.5 Mhz signal is labelled t_clk.
--
-- Sensitivity List: p_clk
--
-- Functions called: none.
-------------------------------------------------------------------------
 divide : process(p_clk)
        variable count: integer:=0;
        begin
           count := count+1;
           if count = 9 then
             t_clk <= '1';
           end if;
           if count = 17 then
             t_clk <= '0';
             count:= 0;
           end if;
         end process divide;

-------------------------------------------------------------------------
-- Process : SETUP
--
-- Description:
-- This process models the initialization procedure the 8031 micro-
-- controller performs to setup the 8253.  The divisor is obtained from
-- the PC.  A full handshake protocol is observed.
--
-- Sensitivity List: empty
--
-- Functions called: none
-------------------------------------------------------------------------
 setup: process
       begin
         wait until reset'event;
         if reset = '1' then
           wait until dr = '1';
           wait until dr = '0';
           hi <= pdata;
           ack <= transport '1' after 1 ns;
           ack <= transport '0' after 21 ns;
           wait until dr = '1';
           wait until dr = '0';
           lo <= pdata;
           ack <= transport '1' after 1 ns;
```

```
          ack <= transport '0' after 21 ns;
          wait until dr = '1';
          wait until dr = '0';
          ack <= transport '1' after 1 ns;
          ack <= transport '0' after 21 ns;
          divisor <= 256 * intval(hi) + intval(lo);
          camcon <= "00000101";
          gate <= '1' after 10 ns;
        else
         gate <= '0' after 10 ns;
        end if;
      end process setup;

-------------------------------------------------------------------------
-- Process : I8253
--
-- Description:
-- This process allows the frame rate to be programmable.  The process
-- models the behavior of an Intel 8253 programmable timer in mode two.
--
-- Sensitivity List: gate,t_clk
--
-- Functions called: none
-------------------------------------------------------------------------
 I8253: process(gate,t_clk)
       variable div: integer:=0;
       variable flag: integer:=0;
       begin
        if t_clk = '1' and t_clk'event and gate = '1' then
          if div = 0 then
            if flag = 1 then
              frame <= transport '1';
              frame <= transport '0' after 20 ns;
            end if;
          div:=divisor;
        else
          div := div - 1;
          flag := 1;
      end if;
    end if;
  end process I8253;

-------------------------------------------------------------------------
-- Process : DAS
--
-- Description:
-- The DAS process models the interrupt handling function of the 8031
-- microcontroller. On the risin edge of r_int, the process accepts
-- the value on t_data provided by the RM_unit.  The process then
-- interrupts the PC and waits for a DR signal.  When the DR signal
-- has been received, the data from the first RMU_SLAVE is read.
-- The PC is interrupted again.  Finally, after the DR signal has been
-- received, the DAS process reads the third RMU_SLAVE value and
-- again interrupts the PC.
--
-- Sensitivity List: empty
```

```
--
-- Functions called: none
-------------------------------------------------------------------------
  DAS: process
       begin
         wait until r_int'event and r_int = '0';
         t_data <= target1;
         p_int <= transport '1';
         p_int <= transport '0' after 20 ns;
         wait until dr = '1';
         wait until dr = '0';
         t_data <= target2;
         p_int <= transport '1';
         p_int <= transport '0' after 20 ns;
         wait until dr = '1';
         wait until dr = '0';
         t_data <= target3;
         p_int <= transport '1';
         p_int <= transport '0' after 20 ns;
         wait until dr = '1';
         wait until dr = '0';
       end process DAS;
-------------------------------------------------------------------------
end behavior;
```

```
-------------------------------------------------------------------
-- Entity: RMU_MASTER
-- Description:
---The RMU_MASTER implements the in-line search algorithm which finds
-- the maximum pixel in each frame.  The RMU_MASTER generates an
-- at the completion of each frame.
-- Input signals:
--              vv : valid video.
--              sclk: synchronized pixel clock
--              ds : digital pixel value
-- Ouput signals:
--              target: address of maximum pixel
--              r_int: interrupt signal
--
-- Generics:
--
--
-------------------------------------------------------------------
use work.cv.all;
entity rmu_master is
 port(vv, sclk : in bit; ds : in bit_vector(7 downto 0);
    target : out bit_vector(10 downto 0);
    r_int : inout bit);
end rmu_master;

architecture behavior of rmu_master is
signal min1,min2,add,count1,count2 : integer;
signal min : integer := 255;
signal count : integer := 0;
signal enable : bit;
signal less : bit := '1';

begin
-------------------------------------------------------------------
-- Process : DONE
--
-- Description:
-- On the rising edge of valid video (vv) signal, the process clears
-- and enables the address counter (count) and sets the interrupt signal
-- (r_int).  On the falling edge of valid video (vv), the process waits
-- one clock period of sclk and then clears the interrupt signal (r_int).
-- The target signal is updated and the current minimum value reset on
-- the falling edge of valid video (vv) as well.
--
-- Sensitivity List: none
--
-- Functions called: binval11
-------------------------------------------------------------------
 -------------------------------------------------------------------
 done: process
      begin
        wait until vv'event;
        if vv = '1' then
          count1 <= 0;
          r_int <= '1';
          wait until sclk='1' and sclk'event;
```

```
          wait until sclk='1' and sclk'event;
          enable <= '1' after 3 ns;
        else
          wait until sclk = '1' and sclk'event;
          wait until sclk = '1' and sclk'event;
          enable <= '0';
          r_int <= '0' after 30 ns;
          min1 <= 255;
          target <= binval11(add);
        end if;
      end process done;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- Process : COUNTER
--
-- Description:
-- The counter process increments on the rising edge of sclk when
-- enable is true.  As the pixels are shifted out of the camera
-- sequentially, the counter represents the addresses of the pixels
--
-- Sensitivity List: sclk
--
-- Functions called: none
-------------------------------------------------------------------------
 counter: process(sclk)
     begin
       if enable = '1' and sclk = '1' then
         count2 <= count + 1 after 25 ns;
       end if;
     end process counter;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- Process : CMP
--
-- Description:
-- The CMP process compares each incoming pixel to the current minimum
-- value. If the incoming pixel is less than the current minimum, the
-- less signal is cleared. This process models one function of the
-- 74LS682 8-bit comparator.
--
-- Sensitivity List: ds
--
-- Functions called: none.
-------------------------------------------------------------------------
cmp: process(ds)
    begin
      if intval(ds) < min and r_int = '1' then
        less <= '0' after 10 ns;
      else
        less <= '1' after 10 ns;
      end if;
    end process cmp;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- Process :  UPDATE
--
```

```
-- Description:
-- On the rising edge of the synchronized pixel clock, the UPDATE process
-- stores the current pixel value as the current minimum value if the
-- less signal is '0'.  The value of the address counter is stored as
-- the position on the array of the minimum value as well.
--
-- Sensitivity List: sclk
--
-- Functions called: intval
-------------------------------------------------------------------------
update: process(sclk)
       begin
         if less = '0' then
            min2 <= intval(ds);
            add <= count;
         end if;
       end process update;
-------------------------------------------------------------------------
     min <= min1 when not min1'quiet else
            min2 when not min2'quiet else
            min;
    count <= count1 when not count1'quiet else
             count2 when not count2'quiet else
             count;
end behavior;
```

```
-------------------------------------------------------------------
-- Entity: RMU_SLAVE
-- Description:
---The RMU_MASTER implements the in-line search algorithm which finds
-- the maximum pixel in each frame.
--
-- Input signals:
--              vv : valid video.
--              sclk: synchronized pixel clock
--              ds : digital pixel value
--              r_int: interrupt generated by RMU_MASTER
-- Ouput signals:
--              target: address of maximum pixel
-- Generics: none
-------------------------------------------------------------------

use work.cv.all;
entity rmu_slave is
 port(vv, sclk : in bit; ds : in bit_vector(7 downto 0);
    target : out bit_vector(10 downto 0);
    r_int : in bit);
end rmu_slave;

architecture behavior of rmu_slave is
signal min1,min2,add,count1,count2 : integer;
signal min : integer := 255;
signal count : integer := 0;
signal enable : bit;
signal less : bit := '1';

begin
-------------------------------------------------------------------
-- Process : DONE
--
-- Description:
-- On the rising edge of valid video (vv) signal the process clears
-- and enables the address counter (count). On the falling edge of
-- valid video (vv), the process waits one clock period of sclk and
-- then clears the counter enable. The target signal is updated and
-- the current minimum value reset on the falling edge of valid
-- video (vv) as well.
--
-- Sensitivity List: none
--
-- Functions called: binval11
-------------------------------------------------------------------
 done: process
      begin
        wait until vv'event;
        if vv = '1' then
          count1 < = 0;
          wait until sclk='1' and sclk'event;
          wait until sclk='1' and sclk'event;
          enable < = '1' after 3 ns;
        else
          wait until sclk = '1' and sclk'event;
```

```
        wait until sclk = '1' and sclk'event;
        enable <= '0';
        min1 <= 255;
        target <= binval11(add);
      end if;
    end process done;

-------------------------------------------------------------------
-- Process : COUNTER
--
-- Description:
-- The counter process increments on the rising edge of sclk when
-- enable is true.  As the pixels are shifted out of the camera
-- sequentially, the counter represents the addresses of the pixels
--
-- Sensitivity List: sclk
--
-- Functions called: none
-------------------------------------------------------------------
counter: process(sclk)
    begin
      if enable = '1' and sclk = '1' then
        count2 <= count + 1 after 25 ns;
      end if;
    end process counter;

-------------------------------------------------------------------
-- Process : CMP
--
-- Description:
-- The CMP process compares each incoming pixel to the current minimum
-- value. If the incoming pixel is less than the current minimum, the
-- less signal is cleared. This process models one function of the
-- 74LS682 8-bit comparator.
--
-- Sensitivity List: ds
--
-- Functions called: none.
-------------------------------------------------------------------
cmp: process(ds)
   begin
     if intval(ds) < min and r_int = '1' then
       less <= '0' after 10 ns;
     else
       less <= '1' after 10 ns;
     end if;
   end process cmp;


-------------------------------------------------------------------
-- Process :  UPDATE
--
-- Description:
-- On the rising edge of the synchronized pixel clock, the UPDATE process
-- stores the current pixel value as the current minimum value if the
-- less signal is '0'.  The value of the address counter is stored as
```

```
-- the position on the array of the minimum value as well.
--
-- Sensitivity List: sclk
--
-- Functions called: intval
-------------------------------------------------------------------------
update: process(sclk)
       begin
         if less = '0' then
            min2 < = intval(ds);
            add < = count;
         end if;
       end process update;
-------------------------------------------------------------------------
     min < = min1 when not min1'quiet else
           min2 when not min2'quiet else
           min;
     count < = count1 when not count1'quiet else
             count2 when not count2'quiet else
             count;
 end behavior;
```

```
-------------------------------------------------------------------
-- Entity: DIGITIZER
-- Description:
-- This entity performs an amplification and inversion of the signal
-- generated by the CAMERA1 entity.  The amplified signal is then
-- digitized using an analog to digital converter.  The converter modeled
-- is the Analog Devices AD9012.
-- Input signals:
--              cs : analog output signal from CCD camera
--              sclk: synchronized pixel rate clock. Used as encode.
-- Ouput signals:
--              ds : digitized version of cs
-- Generics:      none.
--
-------------------------------------------------------------------
use work.all;
use work.cv.all;
entity digitizer is
  port(cs: in real; sclk: in bit; ds: out bit_vector(7 downto 0) );
end digitizer;

architecture behavior of digitizer is
 signal adj_sig : real;
 signal es : bit_vector(7 downto 0);

begin

-------------------------------------------------------------------
-- Process : OP_AMP
--
-- Description:
-- This process models the operation of an inverting op amp.  The gain
-- is set to negative two.
--
-- Sensitivity List: cs
--
-- Functions called: none.
-------------------------------------------------------------------
  op_amp: process(cs)
        begin
          adj_sig <= -2.0 * cs;
        end process op_amp;
-------------------------------------------------------------------
-------------------------------------------------------------------
-- Process : ENCODE
--
-- Description:
-- This process models the analog-to-digital encoding process of
-- the AD9012 converter.  On the rising edge of the synchronized
-- pixel clock, the encode process is begun.  The encoded value
-- is placed on the signal es.
--
-- Sensitivity List: sclk
-- Functions called: binval,convert
-------------------------------------------------------------------
  encode: process(sclk)
```

```
        begin
          if sclk = '1' then
            es <= binval(convert(adj_sig)) after 13 ns;
          end if;
        end process encode;
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- Process : LATCH
--
-- Description:
-- This process places the result of the encode on the ds output signal.
-- The process is activated on the rising edge of the synchronized pixel
-- clock.  However, because of the delay in the encode, the value of
-- placed on the ds output signal is the result of the encode begun on
-- the previous rising clock edge.
--
-- Sensitivity List: sclk
--
-- Functions called: none
-------------------------------------------------------------------------
  latch: process(sclk)
       begin
         if sclk = '1' then
           ds <= es after 13 ns;
         end if;
       end process latch;
-------------------------------------------------------------------------
end behavior;
```

```
-------------------------------------------------------------------
-- Entity: CAMERA1
-- Description:
--- This entity models the Fairchild CCD1500R charge coupled
--  device camera and the CB1000R controller.
-- Input signals:
--              p_clk: pixel rate clock
--              frame: frame rate clock
--              camcon: camera control word
-- Ouput signals:
--              vv : valid video
--              sclk: synchronized pixel rate clock
--              cs : analog output of CCD camera
-- Generics:
--              num_pix: number of pixels in a frame
--              name: file name to read pixel values from
-------------------------------------------------------------------

use work.cv.all;
entity camera1 is
  generic(num_pix : integer; name :string(1 to 8) );
  port(p_clk, frame : in bit;
       camcon: in bit_vector(7 downto 0);
       vv  : inout bit;
       sclk : out bit;
       cs : out real);
end camera1;

architecture behavior of camera1 is
  signal y,temp : pixel;
  signal done : bit := '0';
  signal done1,done2 : bit;
  file cam1: camera_data is in name;
begin
-------------------------------------------------------------------
-- Process : INIT
-- Description:
-- This process simulates the output signals of the controller. The
-- process waits for the rising edge of the frame clock.  Upon its
-- receipt, if camcon equals 05 hex, the vv signal is asserted.  The
-- process then waits for the rising edge of the done flag to clear the
-- vv signal.
-- Sensitivity List: empty
-- Functions called: none

  init : process
         begin
           wait until frame = '1' and frame'event;
           wait until p_clk = '1' and p_clk'event;
           if camcon = "00000101" then
             vv <= transport '1';
             done1 <= '0';
           end if;
           wait until done = '1';
           vv <= transport '0';
```

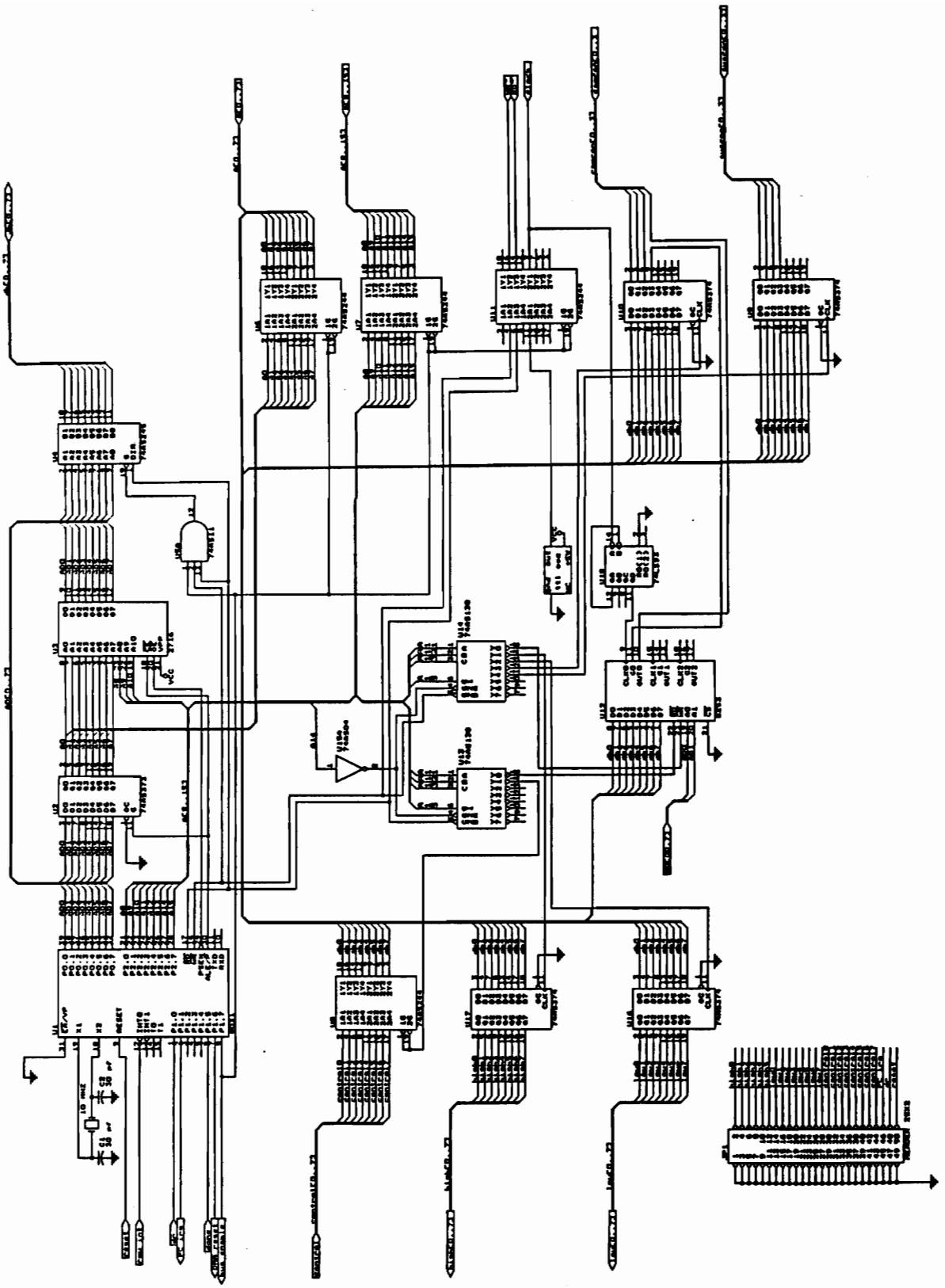
```
        end process init;
-------------------------------------------------------------------------

-------------------------------------------------------------------------
-- Process : scan
-- Description:
-- This process simulates the operation of the CCD camera.  If valid
-- video (vv) is asserted, the camera reads a pixel value from a data
-- file on the rising edge of the pixel rate clock.  The pixel rate
-- clock is synchronized with the ouput of the value read from the
-- data file.
-- Sensitivity List: p_clk
-- Functions called: a_to_r

  scan : process(p_clk)
        variable x: pixel;
        variable counter : integer:=0;
        begin
          if(vv = '1' and p_clk = '1') then
            if ENDFILE(cam1) then
              cs <= 0.0;
              counter := counter + 1;
            else
              read(cam1,x);
              cs <= a_to_r(x);
              counter := counter + 1;
          end if;
            if counter = num_pix then
              counter := 0;
              done2 <= '1';
            end if;
          else
            if vv = '0' then
              cs <= 0.0;
             end if;
          end if;
   sclk <= p_clk;
 end process scan;
-------------------------------------------------------------------------

    done <= done1 when not done1'quiet else
          done2 when not done2'quiet else
          done;
end behavior;
```

```
package CV is
  subtype pixel is string(9 downto 1);
  subtype tadd is string(12 downto 1);
  type camera_data is file of pixel;
  type tfile is file of tadd;
  type tstr is array(1 to 3) of tadd;
  function a_to_r(input : pixel) return real;
  function binval11(val:integer) return bit_vector;
  function binval(val:integer) return bit_vector;
  function intval(val: bit_vector) return integer;
  function convert(analog: real) return integer;
  function t_to_s(input: bit_vector; t: integer) return tadd;

end CV;
package body CV is
-------------------------------------------------------------------------
   function a_to_r(input:pixel) return real is
   variable sum :real;
   variable i,j: integer;
   begin
     sum := 0.0;
     j := 1;
     for i in 9 downto 1 loop
       j:=j-1;
       if input(i) = '.' or input(i) = LF then
         j:=j+1;
       else
         if input(i) = '1' then sum := sum + 10.0**j;
         elsif input(i) = '2' then sum:=sum+2.0*10.0**j;
         elsif input(i) = '3' then sum:=sum+3.0*10.0**j;
         elsif input(i) = '4' then sum:=sum+4.0*10.0**j;
         elsif input(i) = '5' then sum:=sum+5.0*10.0**j;
         elsif input(i) = '6' then sum:=sum+6.0*10.0**j;
         elsif input(i) = '7' then sum:=sum+7.0*10.0**j;
         elsif input(i) = '8' then sum:=sum+8.0*10.0**j;
         elsif input(i) = '9' then sum:=sum+9.0*10.0**j;
         elsif input(i) = '0' then sum:=sum;
         end if;
       end if;
     end loop;
     return sum;
   end a_to_r;
-------------------------------------------------------------------------
  function binval(val:integer) return  bit_vector is
   variable vect: bit_vector(7 downto 0);
   variable temp1,temp2,a : integer;
   variable N: integer;
  begin
    temp1 := val;
    for N in 7 downto 0 loop
      temp2 := temp1 / (2**N);
      temp1 := temp1 - temp2*(2**N);
      if temp2 = 1 then
        vect(N) := '1';
      else
        vect(N) := '0';
```

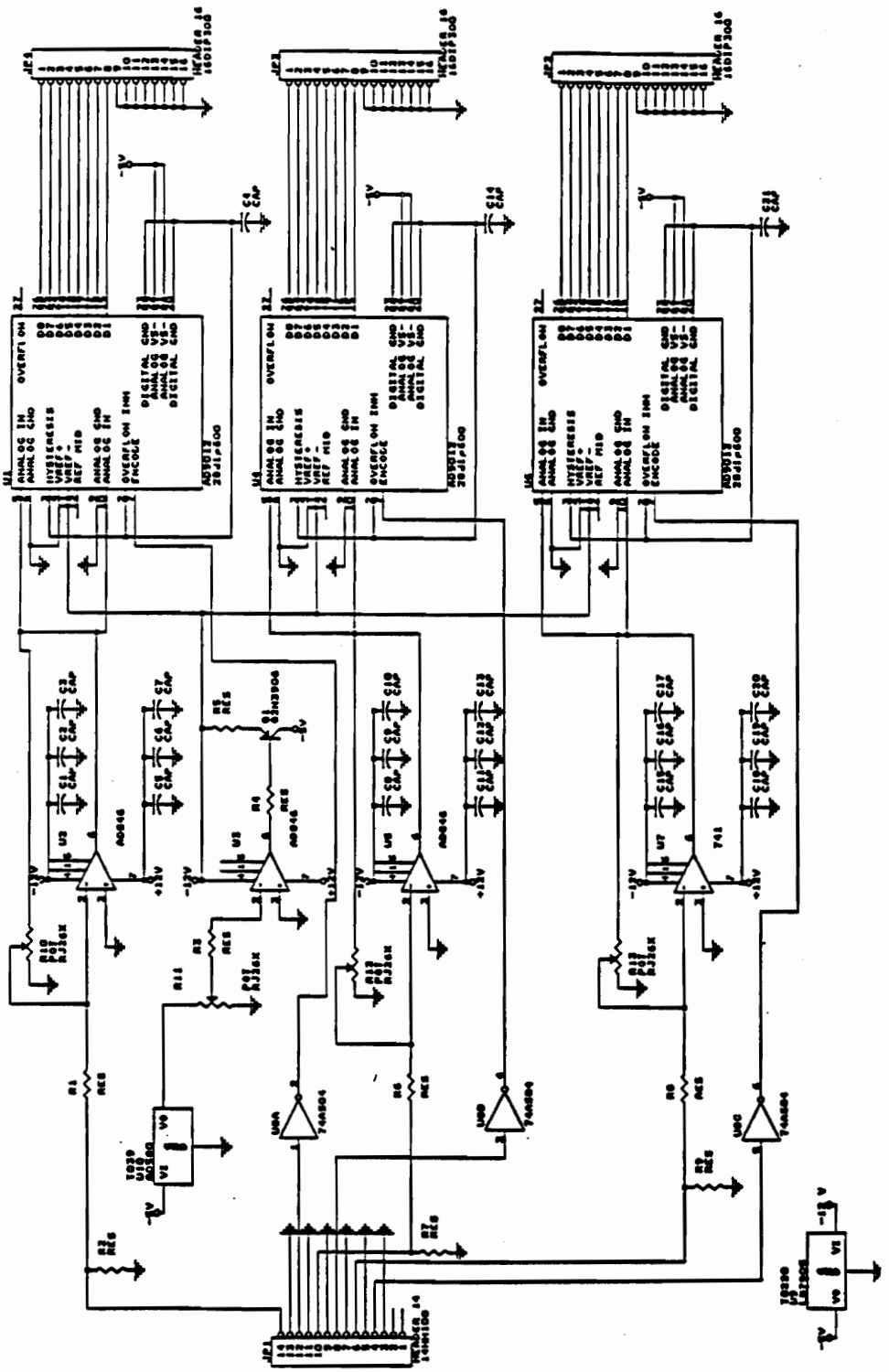
```
    end if;
   end loop;
   return vect;
  end binval;
-------------------------------------------------------------------------
  function binval11(val:integer) return  bit_vector is
   variable vect: bit_vector(10 downto 0);
   variable temp1,temp2,a : integer;
   variable N: integer;
  begin
    temp1 := val;
    for N in 10 downto 0 loop
      temp2 := temp1 / (2**N);
      temp1 := temp1 - temp2*(2**N);
      if temp2 = 1 then
        vect(N) := '1';
      else
        vect(N) := '0';
      end if;
    end loop;
    return vect;
  end binval11;
-------------------------------------------------------------------------
   function intval(val:bit_vector) return integer is
    variable sum: integer:=0;
   variable N: integer;
  begin
    for N in val'LOW to val'HIGH loop
      if val(N) = '1' then
        sum := sum + (2**N);
      end if;
    end loop;
    return sum;
  end intval;
-------------------------------------------------------------------------
  function convert(analog: real) return integer is
    constant res : real := -7.813e-3;
    variable j,i : integer;
    variable temp : bit_vector(7 downto 0);
     variable atemp,k: real;
  begin
    j := 0;
    k:=255.0;
    for i in 255 downto 0 loop
      atemp := k * res;
      if(analog > atemp)  then
        j:=j+1;
      end if;
     k:=k-1.0;
   end loop;
   return j;
  end convert;
-------------------------------------------------------------------------
  function t_to_s(input:bit_vector; t: integer) return tadd is
  variable i,j: integer;
  variable out_s : tadd;
```

```
 begin
   for i in 12 downto 2 loop
     if input(i-2) = '1' then
       out_s(i) := '1';
     else
       out_s(i) := '0';
     end if;
   end loop;
  if t = 3 then
    out_s(1) := LF;
   else
    out_s(1) := ' ';
   end if;
  return out_s;
 end t_to_s;

end cv;
```

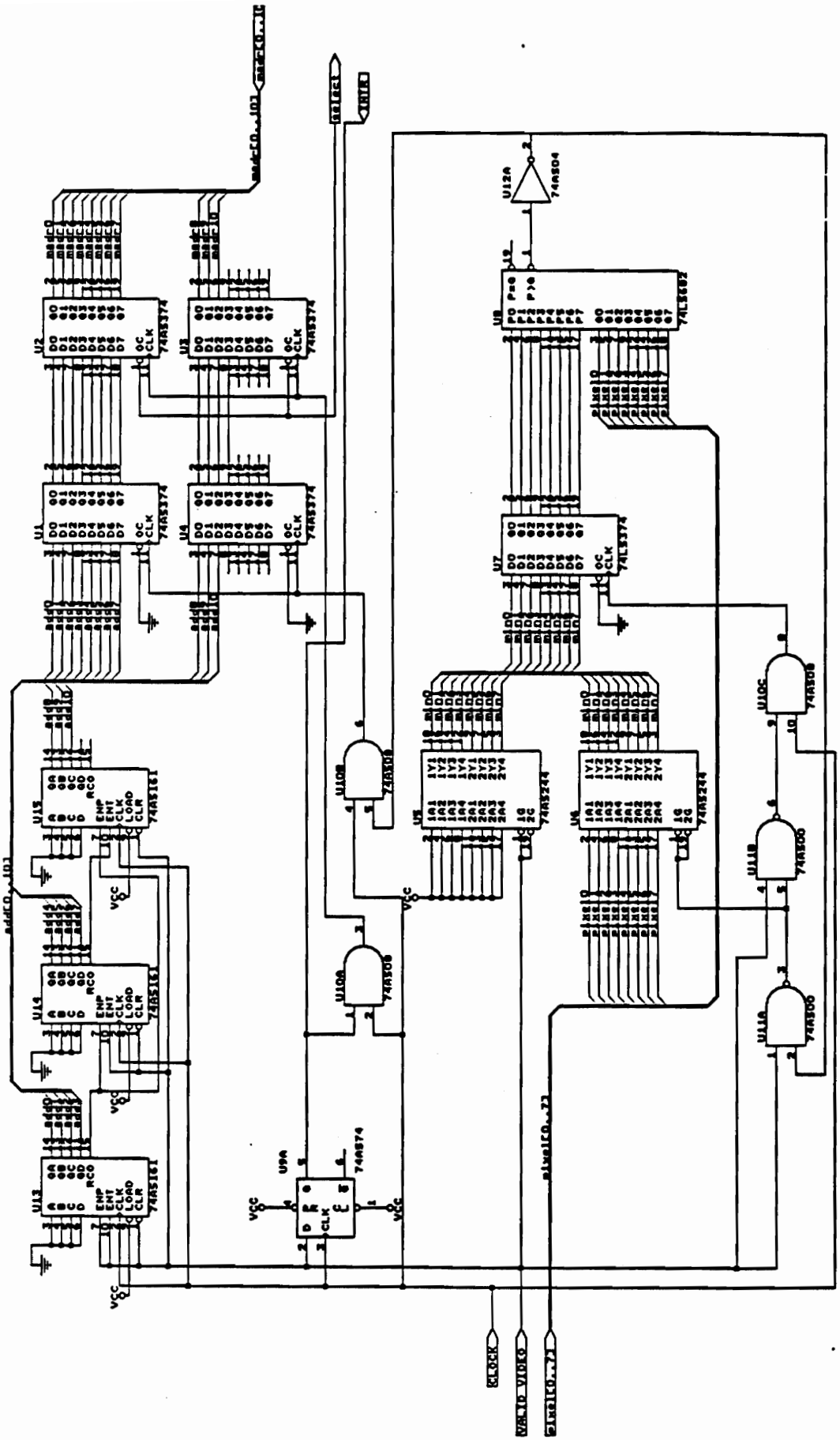
Appendix B. Schematics



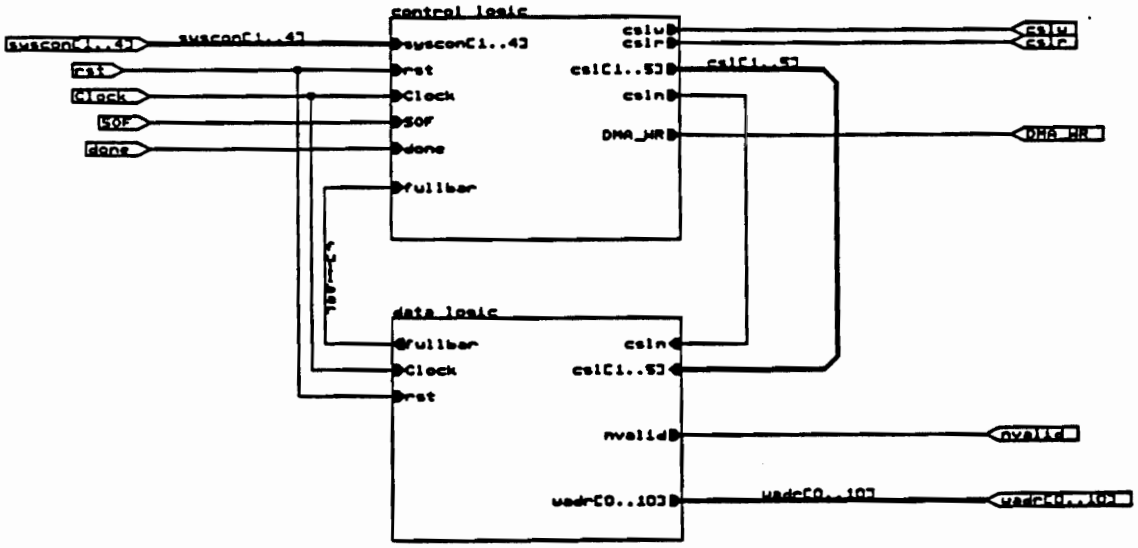
System control unit



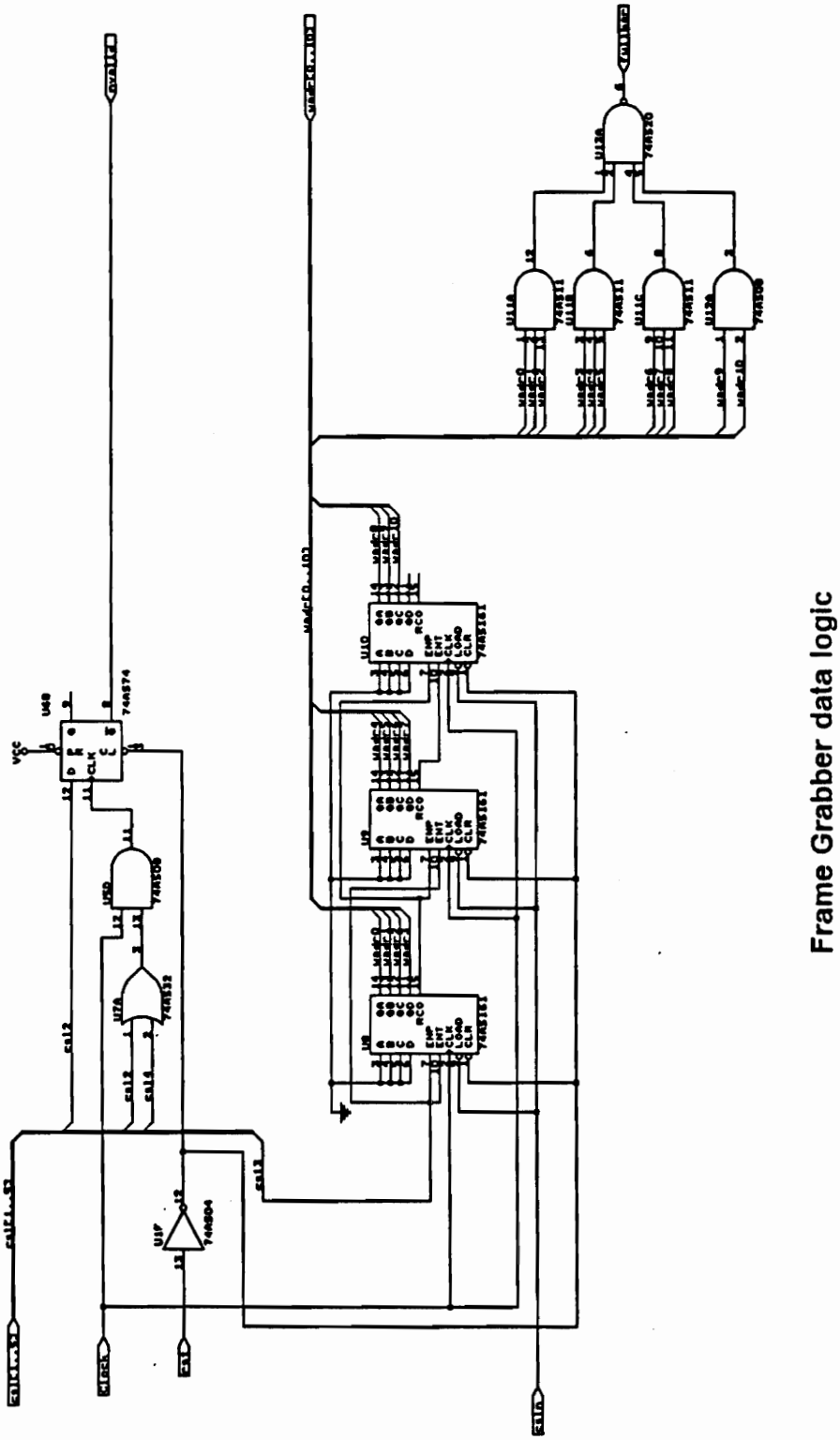
Analog-to-Digital conversion logic



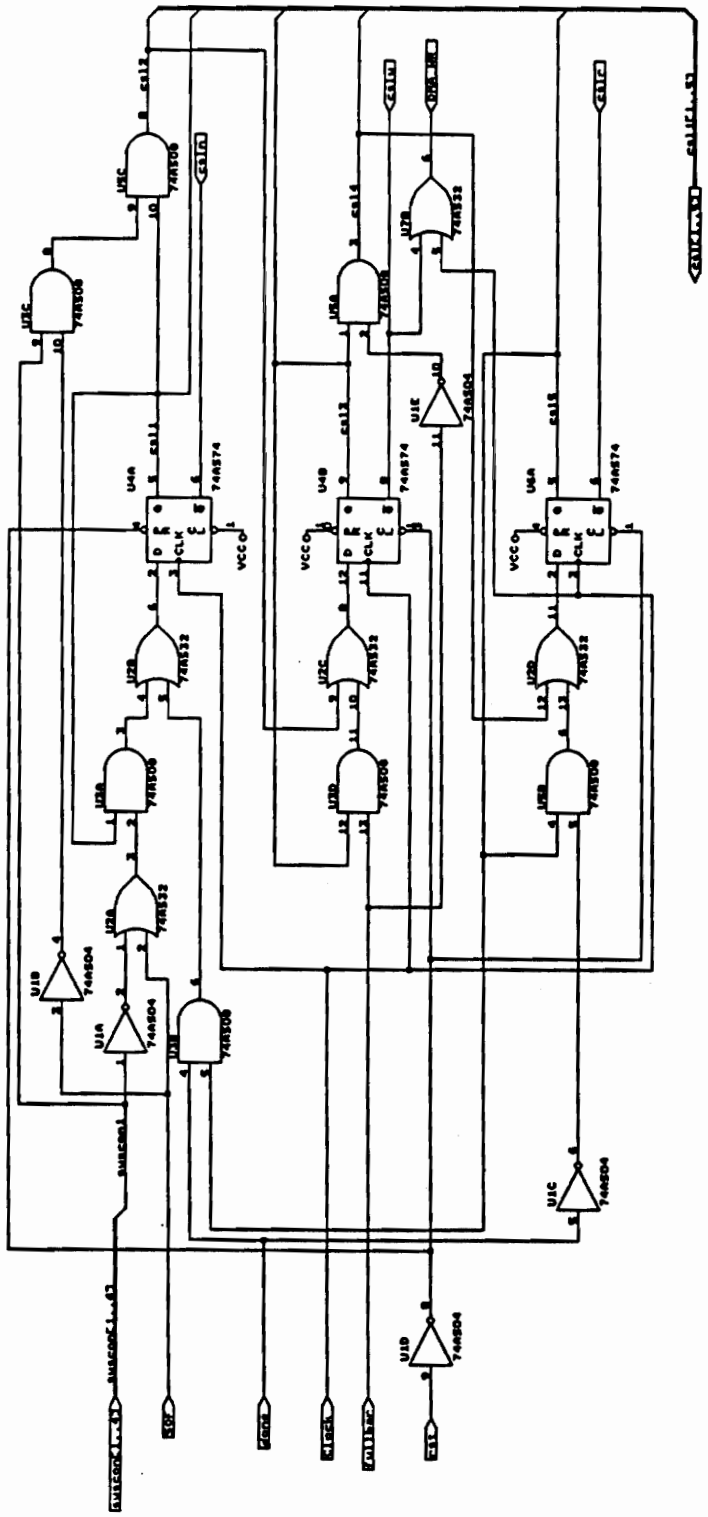
Running maximum unit



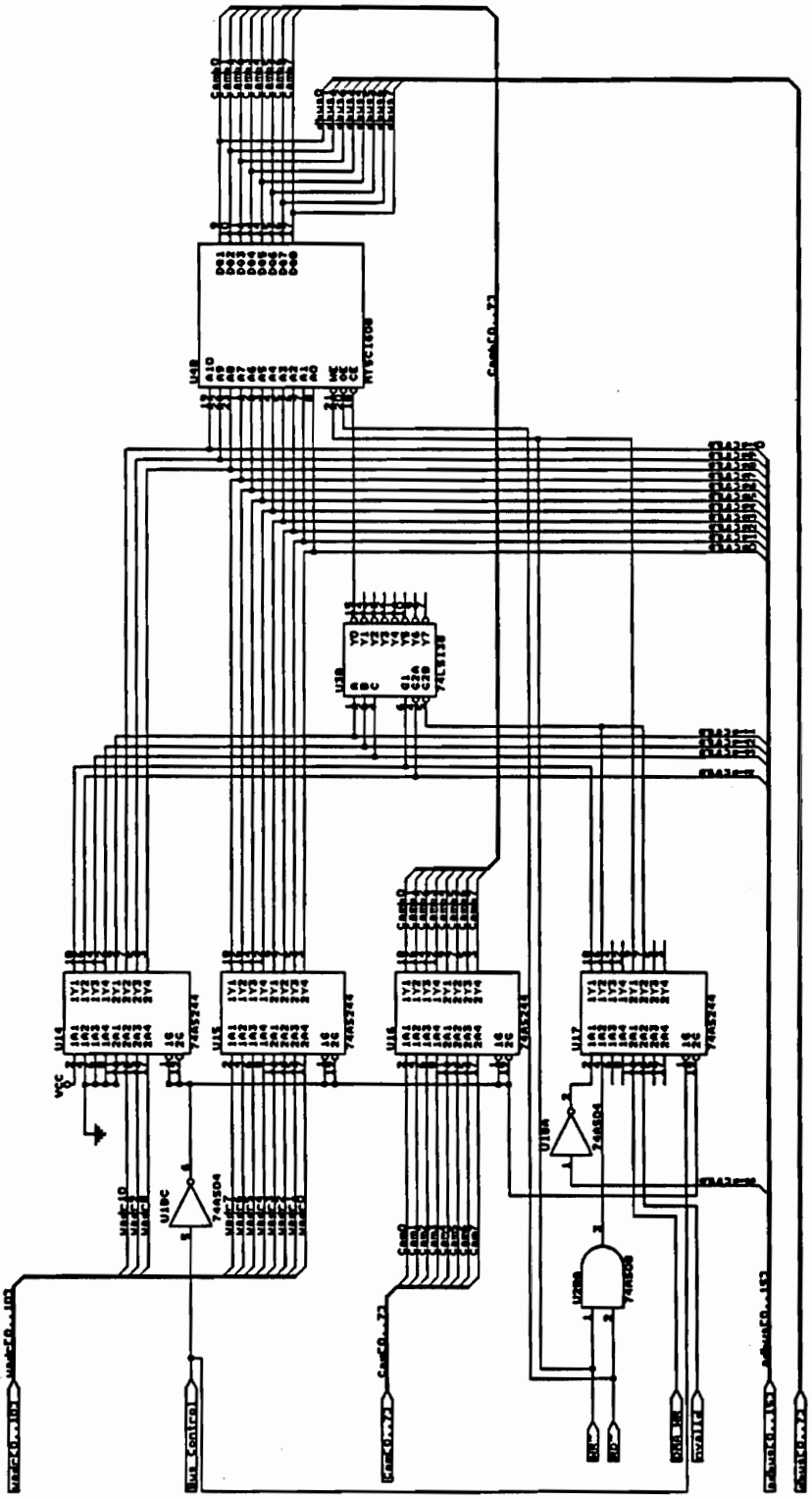
Frame Grabber block diagram



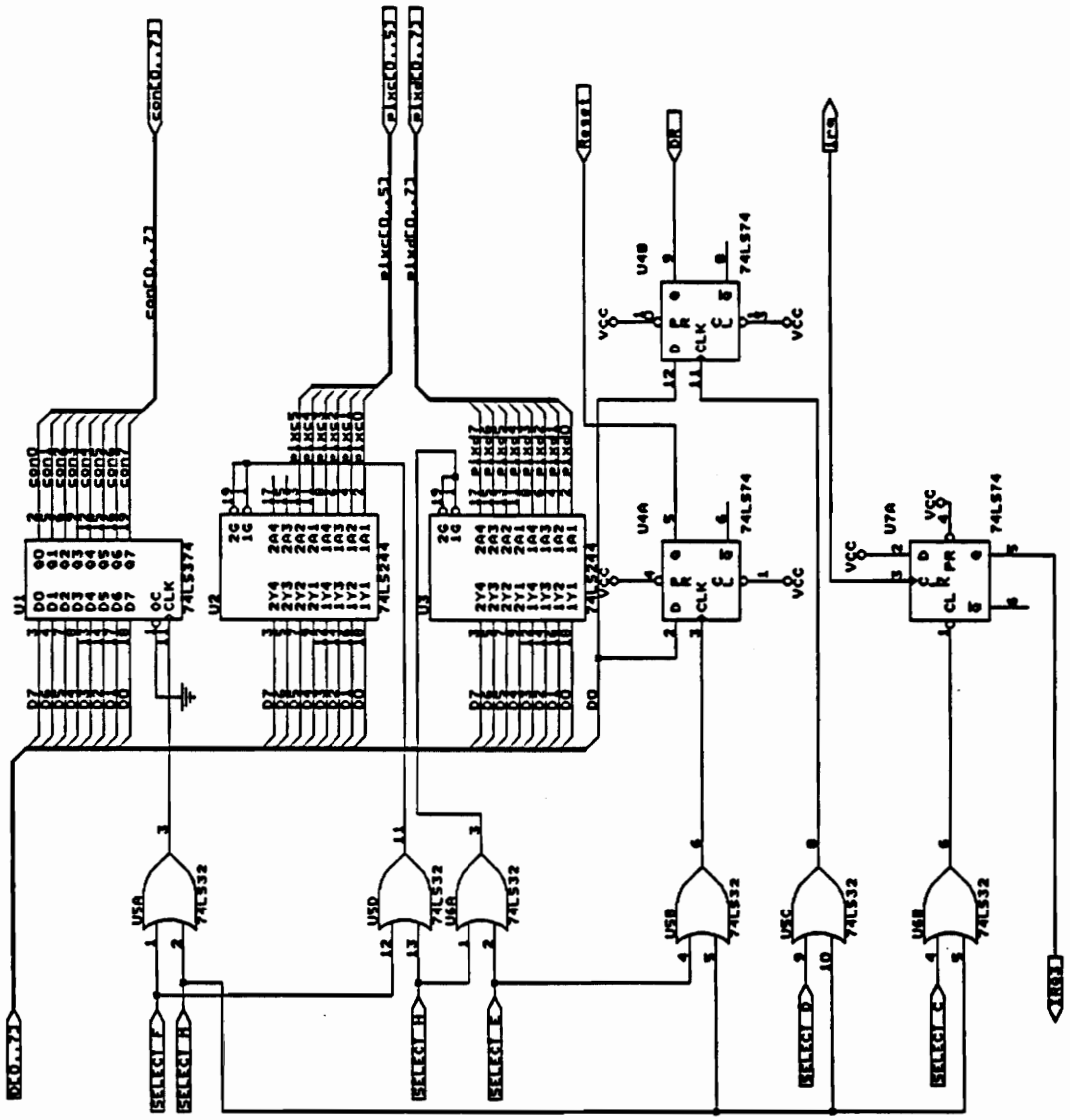
Frame Grabber data logic



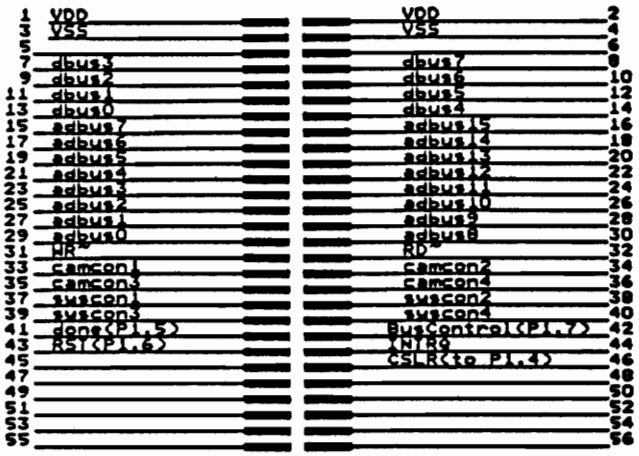
Frame Grabber control logic



Frame Grabber memory



PC Interface logic



STD-BUS assignment

Vita

Michael Anestis Colbert was born in Rhodes, Greece, on April 15, 1966. He received the B.S. degree in electrical engineering from the Virginia Polytechnic Institute and State University, Blacksburg, Virginia in 1988. He is currently completing the M.S. degree in electrical engineering at the same institution.

Mr. Colbert was a laboratory technician in the ESM Vibrations Laboratory at Virginia Tech. His work involved the development of measurement systems and laboratory apparatus. He will begin work at the Naval Research Laboratory upon completion of the M.S. degree. His research interests include digital design and computer engineering.

Mr. Colbert is a member of the Institute of Electrical and Electronics Engineers, Tau Beta Pi, and Eta Kappa Nu.