A Stereo Vision System with Automatic Brightness Adaptation

by

Keith G. Fife

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of

Master of Engineering
and
Bachelor of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

May 1999
June 1999

Signature of Author _____
Department of Electrical Engineering and Computer Science
7 May 1999

Certified by _____
Charles G. Sodini, Ph.D.
Professor of Electrical Engineering
Thesis Supervisor

Accepted by _____
Arthur C. Smith, Ph.D.
Professor of Electrical Engineering
Graduate Officer

A Stereo Vision System with Automatic Brightness Adaptation

by

Keith G. Fife

Submitted to the Department of Electrical Engineering and Computer Science
on 7 May 1999 in partial fulfillment of the requirements for the degrees of
Master of Engineering
and
Bachelor of Science in Electrical Engineering and Computer Science

Abstract

This thesis describes the development of an automatic brightness adaptive imaging system for use in stereo vision algorithms implemented for a variety of processing architectures. A 256×256 array of wide dynamic range pixels with on-chip A/D converters provides the digital data path for a feedback network which controls the charge integration parameters at each pixel. The first goal of the project was to build a real-time demonstration of the imager with configurable compression functions. Secondly, electronic irising was employed by controlling the global charge integration time based on the average intensity of the image. In addition to electronic-irising, the imaging system employs a linear or a logarithmic compression scheme based on the image data. The controller fits the compression function to the image by comparing the average intensities of many different regions within the image. Finally, a 3-camera stereo-vision system was developed with data transfer to a PC through the PCI bus at 60fps. The imagers are synchronized and controlled based on the center imager's data which allows for consistent object correlation in stereo vision algorithms.

Thesis Supervisor: Charles G. Sodini
Title: Professor of Electrical Engineering

Acknowledgments

I am very grateful to the number of people who have contributed to the successful completion of my master's project.

First, I'd like to extend my gratitude to Professor Charles Sodini for providing me with this research opportunity. Throughout the project he exhibited trust in my judgment and confidence in my ability. I also thank Dr. Ichiro Masaki for his insight and judgment in this technical field.

Without the help of Zubair Talib I would not have been able to meet the deadline for the first stage of this project. He also helped me out with general computing issues and warmed me up to the idea of using PCs rather than UNIX Workstations when the proper tools fit better there. He also provided many useful tools such as perl programs and LaTeX documents. Thanks to Lane Brooks for being the key player in developing the software for displaying the 3 real-time images on the PC.

I thank all of the students and staff in Professor Sodini's research group. Many thanks to Steven Decker for his help on the first stage of the project and in general for creating the CMOS imager chip upon which this thesis is based. Pablo Acosta-Serafini allowed me to move the lab into our office and lengthen my desk with the infamous desk extender. Jim MacArthur offered assistance and advice with design and general engineering problems. Many thanks are in order to Patricia Varley and Anne Hunter for their generous assistance in dealing with MIT administrative tasks.

Finally, I thank my family. In addition to all her support, and despite her busy schedule, my wife kindly proof-read my thesis. I also thank my Mom and Dad. It was they who encouraged me to explore my scientific interests while growing up. I'm very grateful for their constant support, encouragement, and money.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The technology of microelectronic devices like sensors and digital processors has reached a stage where complex intelligent vehicle control is now feasible [1], [2], [3], [4]. Many practical image processing algorithms created for this application have been designed to operate on digital processors in real-time using conventional imagers [5], [6]. Improvements in this field will continue to be made as image sensors evolve and as processing power increases.

One limitation in image sensors is the effective dynamic range. Imaging applications often deal with situations in which lighting conditions are far from optimal. In particular, these may include objects positioned against strong back lighting which causes the object's details to become too dark if the camera adjusts itself to the high average brightness. In some situations there may be many regions with steep gradations of brightness which are hard to handle by standard cameras. Other situations depend on the dynamic behavior of the camera. If there are abrupt changes in illumination, the camera may not be capable of effectively readjusting its parameters.

Since the power of computation continues to improve, there has been a large motivation to use general purpose processors in intelligent vehicle applications. These processors are generally well documented, easy to use and compatible with previous architectures of the same type. While the architecture of these processors may not be the most efficient for low-level image processing, algorithms have been developed that successfully operate in real-time on these processors [7], [8]. More conventional image processing algorithms like edge-detection, smoothing and segmentation, median filtering and optical flow are generally computationally intense but are based on repeatedly performing a few relatively simple operations to every pixel. The pixel-parallel architectures described in [9] and [10] have been created for such processing.

Improvements in imaging can be made with wide dynamic range algorithms implemented in hardware and software for various processing systems. This thesis is based on a CMOS imager chip developed by Decker and Sodini that has a controllable function for charge integration [11]. Both the amount of time that the pixel reacts to light and the level to which it can integrate charge can be dynamically controlled.

While research in intelligent vehicles has been largely based on driver assisted schemes like adaptive cruise control, lane following, and collision avoidance, the research can also be applied to fully autonomous systems. Stereo vision in particular has been a necessary component in nearly all passive vision systems [12], [13]. Real-time stereo vision with

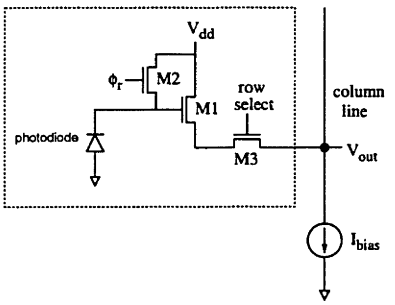

Figure 1.1: 3-transistor voltage output pixel.

brightness adaptation could make both driver assisted and fully autonomous vehicles more reliable.

The goal of this project is to design and build a stereo vision system that is well suited for the image processing tasks of an intelligent vehicle application. To support the proposed machine vision system, first a description of general imager architecture is presented, followed by the specific requirements of imagers used for machine vision. Finally a 3-camera automatic brightness adaption system is presented.

1.1 Imager Architecture

An imager is an array of pixels that convert light into charge, current or voltage. The two main technologies for solid-state imagers are charge-coupled device (CCD) arrays and metal-oxide semiconductor (MOS) arrays. One advantage of MOS imagers is that they can be built into standard Complementary Metal Oxide Semi-conductor (CMOS) processes which allow easier integration with digital and analog circuitry. A CMOS array of pixels which uses on-chip circuitry to extend the dynamic range of an image is used for this project.

1.1.1 3 Transistor MOS pixel

An MOS imager consists of any pixel array which uses MOSFET transistors to convey the signal from the pixels to the output circuitry. The basic form of the *voltage* readout pixel is shown in Figure 1.1. The pixel is reset by pulling ϕ_r high. In a *hard* reset, ϕ_r is pulled more then a threshold drop above V_{dd} and the photodiode is reset to V_{dd}. In a *soft* reset, ϕ_r does not go high enough to equalize the voltage across M2. Assuming ϕ_r goes to V_{dd}, the photodiode is reset to approximately $V_{dd} - V_{te,2}$. Integration starts when ϕ_r goes low. The photodiode potential drops at a rate proportional to the illumination. The effect comes from the *photocurrent* in the diode discharging a parasitic capacitor. At the end of the integration period, the row select device M3 turns on, connecting M1's source to the current source at the bottom of the column line. The output signal is the voltage on the column line, which has an approximate linear dependence on the illumination.

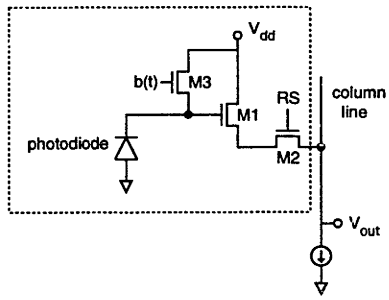

Figure 1.2: Wide dynamic range pixel.

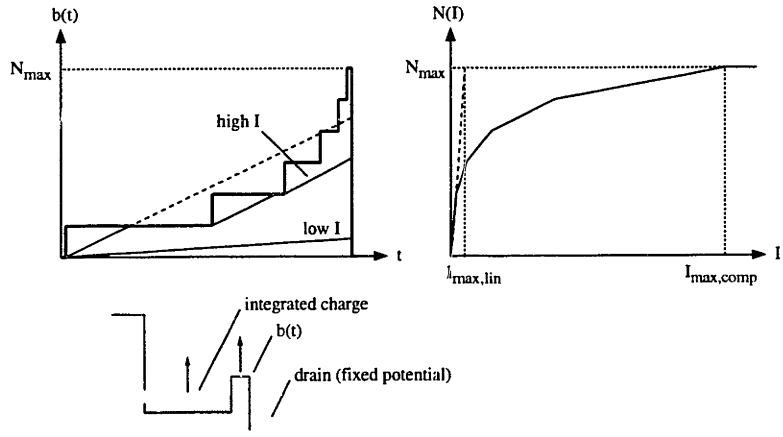

Figure 1.3: Stepped logarithmic barrier function.

1.1.2 Wide-Dynamic Range MOS pixel

While conventional imagers frequently employ a mechanical or electronic shutter to adjust the global integration time, the wide-dynamic range imager by Decker contains a pixel with a lateral overflow drain as shown in Figure 1.2. The voltage level on the gate of M3 can be varied during the integration period to control the amount of charge that is accumulated.

The function $b(t)$ applied to this gate is referred to as a barrier function because the amount of charge a pixel can accumulate over a given period is limited by the voltage level on this gate. Figure 1.3 shows an example of a barrier compression function which can be applied to this gate. The imager effectively uses a long integration period for regions of low illumination and a short integration period for regions of high illumination. The improvement in dynamic range demonstrated by this functionality is shown by the values of Imax,lin and Imax,comp.

The imager is capable of delivering over 300 frames/sec and has a column-parallel architecture. The floorplan of the imager is shown in Figure 1.4.

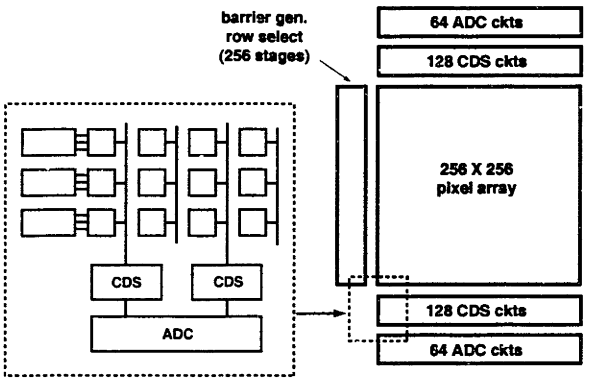

Figure 1.4: Imager architecture.

1.2 Imager System Requirements

The main criteria for imagers used in intelligent vehicles is the frame rate, dynamic range and pixel array size. Some of the requirements for the resolution and accuracy of imagers used for autonomous mobility have been outlined by Kelly in [14]. The frame rate necessary for the system depends on the speed of the vehicle or the intelligent machine. Most systems could use a variable frame rate in the range from 5 to 100 fps. The NTSC standard of 30 fps is commonly used, although not always sufficient. The dynamic range required in most applications is greater than that of linear imagers. Since image intensities may vary by over six orders of magnitude, details in the darker regions of an image are lost by most conventional imagers. While auto-irising may eliminate the saturation of bright objects in a scene, the dynamic range captured by the imager remains unchanged. Finally, the pixel array size or spatial resolution requirements for imagers is usually very large. However, many systems use multiple cameras with different focal lengths to capture information at a number of discrete distances from the vehicle. Therefore, in multiple imager systems, lower spatial resolution imagers are sufficient.

1.3 Automatic Brightness Adaptation

Automatic brightness adaptation includes both electronic irising and automatic dynamic range adjustment. In scenes that do not have a large dynamic range, a linear mapping of the pixel's brightness level should be used because features in an image are more distinguishable when uncompressed. However, when the scene has a large dynamic range, a logarithmic compression may be preferable. The imager should be able to control its image compression scheme based on the dynamic range of the image. In addition, it must control the global integration time based on the average brightness of the entire image. Furthermore, for an imaging system that contains multiple imagers used for stereo vision, all imagers should receive the same compression and integration parameters.

1.4 Project Description

The CMOS imager by Decker presents an array of pixels that can meet many of the performance requirements described in Section 1.2. In order to create a useful imaging system for intelligent vehicle applications, the system must provide automatic brightness adaptation. The pixel array must be controllable from the host system and the output format must ordered in such a way that the data can be readily processed. The automatic brightness adaptive system will continually provide the processor with useful image data, even over the most diverse conditions. It is the goal of this thesis to produce a 3-camera stereo vision system with the specifications listed below:

- Automatic brightness adaptation (control range from 70db to 90db)
- High frame rate (60 fps)
- PCI interface for general purpose processing
- Secondary digital data path for customized processor
- NTSC output for viewing on a standard monitor
- Compact size

In the 3-camera system, all the imagers are synchronized and adjusted based on the center imager's data content. This allows for consistent object correlation in a stereo algorithm. Some of the challenges of this project include:

- Creating a small printed circuit board to drive the imager array and decode uploaded compression functions.
- Creating a protocol which uses a small number of interconnections and allows multiple imagers to be synchronized.
- Designing an algorithm to control the global integration time and to actively switch compression schemes based on the incoming imager data.
- Building a PCI interface which allows three 256×256 images at 60ps to be loaded and displayed on a PC with minimal latency and continuous throughput.
- Understanding both the analog and digital issues associated with the imager array in order to produce high-quality images.

1.5 Thesis Organization

This chapter has described the motivation for research in automatic brightness adaptation for intelligent vehicle control applications. Chapter 2 describes how to use the wide dynamic range pixel and the architecture of the CMOS imager. Chapter 3 discusses alternative brightness adaptation systems. Chapter 4 presents the various design stages of

the imager system. Chapter 5 describes the implementation and results of the automatic brightness adaptation circuit boards. Chapter 6 summarizes the accomplishments and offers suggestions for future work. Appendices A and B document the NTSC and digital imager boards. Appendix C describes the software system and interface for the real-time demo on a PC.

Chapter 2

Imager Architecture and Compression Functions

The main topic in this chapter is the functionality of the wide dynamic range pixel. A detailed discussion of the layout and circuit design involved in the 256 × 256 pixel array is found in [11]. The two parameters that may be adjusted in the barrier function are the voltage level and time. The following sections describe the pixel array and the motivation behind adjusting these parameters during the frame period.

2.1 The Pixel Array

The micro-chip used for this project is shown in Figure 2.1. The chip features a 256 × 256 array of wide dynamic range pixels that can be controlled from the Pin Grid Array (PGA) package. The imaging array also contains the CDS and ADC circuits necessary for the data conversion.

The circuit for the wide dynamic range pixel is represented in Figure 2.2. The transistor M4 is the overflow gate used to control the compression function. The voltage on this gate decreases in steps during the integration period. The chip allows for 8 different stepped

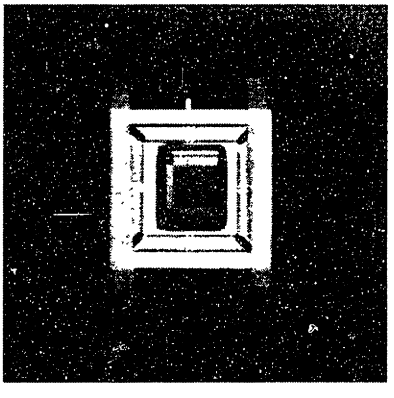

Figure 2.1: The 256 × 256 CMOS pixel array.

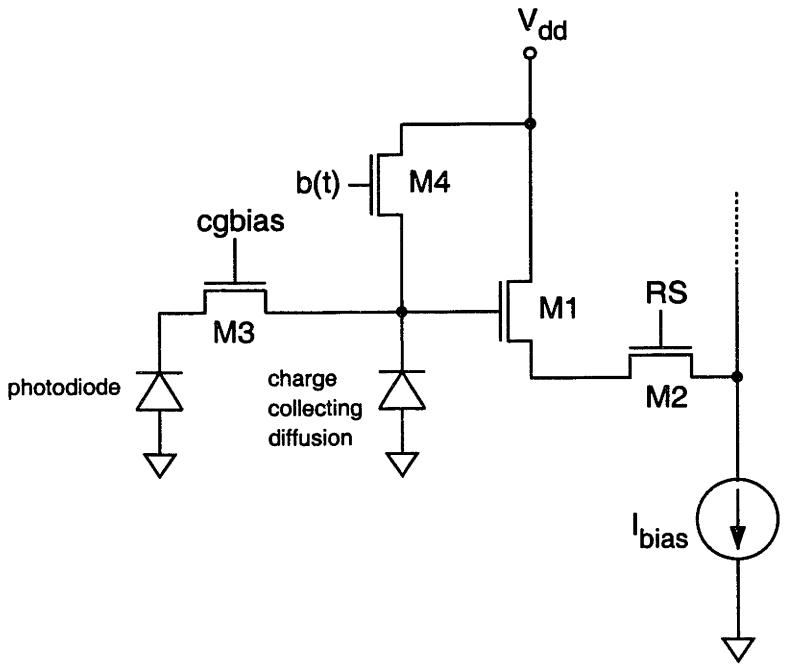

Figure 2.2: Circuit for wide dynamic range pixel.

voltage levels. Depending on the location and timing of these steps, many compressive characteristics can be approximated. M3 is the charge spill gate which increases the sensitivity of the pixel because it acts as a common gate amplifier. The photocurrent flows into the low impedance source node and is discharged into the high impedance drain. This allows charge collected by the large photodiode to be sensed by the small charge collection diffusion. This feature was actually disabled for reasons described in Chapter 5. The source follower M1 buffers the pixel from the column line loading. M2 is the row select transistor that connects the source follower output to the column line during the row read-out period.

The integration period starts with the lateral overflow gate and sense diffusion at the highest potential. Charge generated in the large photodiode diffuses across M3 into the sense diffusion which lowers its potential. M4 applies a time-varying potential *barrier* to the electron flow into the charge collecting diffusion. At the end of the integration period, M2 turns on, allowing a bias current to flow through the source follower device. After voltage on the column line has settled, the pixel output voltage is sampled by the correlated double sampling (CDS) circuit. The first sample represents the amount of charge the pixel has accumulated during the integration period plus the amount it started with before the integration period started. Next, M4's gate is pulled to its maximum voltage which resets the pixel. The pixel output voltage is then sampled again. The difference between the two samples represents the total amount of light that was converted to charge during the integration period. After M4 resets the pixel, its gate drops to the next step voltage level and starts the integration period again.

2.2 The Compression Function

A wide dynamic range image is one in which the ratio of light intensity between the darkest and brightest portions of the image is more than what can be reliably captured by a standard camera. The functionality of the lateral overflow transistor in the pixel array allows the imager to capture and quantify high levels of illumination that would otherwise saturate the pixel. Figure 2.3 shows both linear and logarithmic compression functions that can be applied to the gate during one frame period.

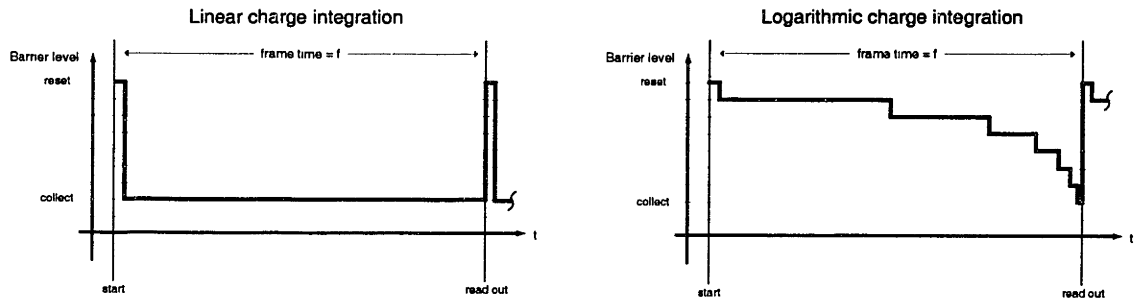

Figure 2.3: Linear and logarithmic compression functions.

For the linear charge integration function, the *barrier level* is used only to reset the pixel. When it falls to the lowest level, the pixel level is always above the barrier unless the pixel saturates. The eight levels in the pseudo logarithmic compression function cause different illumination levels on the pixel to be mapped linearly into 7 regions of compression. The first region is the largest and handles the lowest illumination levels. The regions become progressively smaller until the last region, which represents the brightest levels of illumination. In essence the low illumination levels are emphasized and the high illumination levels are compressed. This is similar to the characteristics produced when taking the logarithm of illumination. Figure 2.4 shows five steps in the barrier level where reset has been referenced to the lowest voltage rather than the highest voltage. This convention allows a more straight-forward transition into the charge versus illumination plot in the same figure. The five regions of compression are represented by the five straight line segments which approximate the logarithmic function. In the *Collected Charge vs. Illumination* plot, I_1 represents the slope of the line passing through the points $(0,0)$ and (t_1, b_1) in the *Barrier Voltage vs. Integration Time* plot. In general, each of the I_x points are the slopes of lines representing the maximum light intensity that is unaffected by the *barrier level* during that step of the function.

Nearly any compression characteristic can be implemented by varying the lateral overflow gate during the frame period. Four different compression characteristics are shown in this section by applying the functions to the pixel array during the capture of the scene shown in Figure 2.5. The scene requires a wide-dynamic range because in order to read the words "MIT" near the light source the manual iris on the lens must be opened to the level shown in the first image. However, In order to see the details in the light, it is necessary to close the iris to the point where the word becomes unreadable.

The first compression function shown in Figure 2.6 gives the largest dynamic range. The barrier is held at the first step voltage for nearly the entire integration period. This allows

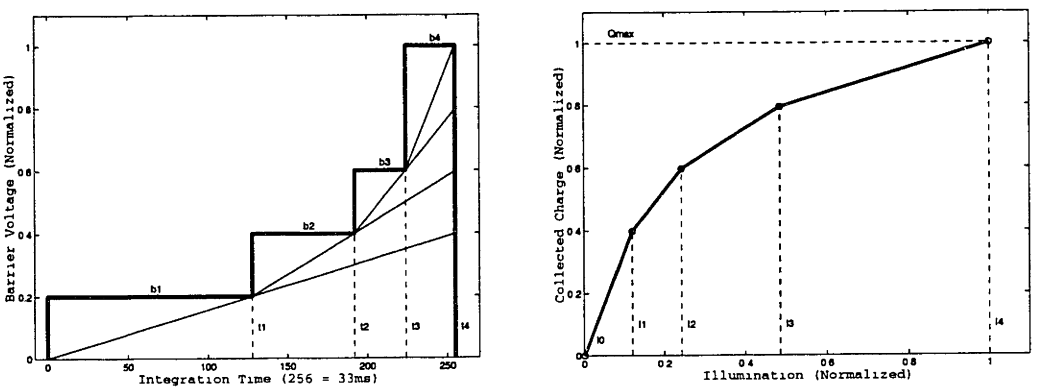

Figure 2.4: Barrier function with charge versus illumination plot.

only the lowest levels of light to be mapped linearly to the voltage readout circuitry. Just before the end of the integration period, the barrier moves to the top step which allows only the very high intensities to reach the upper levels of the quantization. Although this will give the largest dynamic range, the mid range light levels will be lost. In order to get a better dark response, the first step of the barrier voltage can be moved to nearly half-scale as shown in Figure 2.7. By adding an intermediate step between the low and high levels, the image in Figure 2.8 is produced. This turns out to be the best visual compression function for this scene. However, the best compression function to apply to a general set of wide-dynamic range scenes is shown in Figure 2.9. By using all 8 barrier levels, a wide dynamic-range image is produced with transition points that are not noticeable. The details in the light are not as clear in the image because the dynamic range is about 4 times less than that in Figure 2.6.

The previous plots of the collected charge versus the photocurrent have been scaled appropriately based on the measured data of the imager. In order to determine the low end of the dynamic range, the dark response and the random noise were used. The photocurrent estimates were made based on the saturation level and the relationship between photocurrent and output level. Table 2.1 gives a summary of the imager's performance taken from [11].

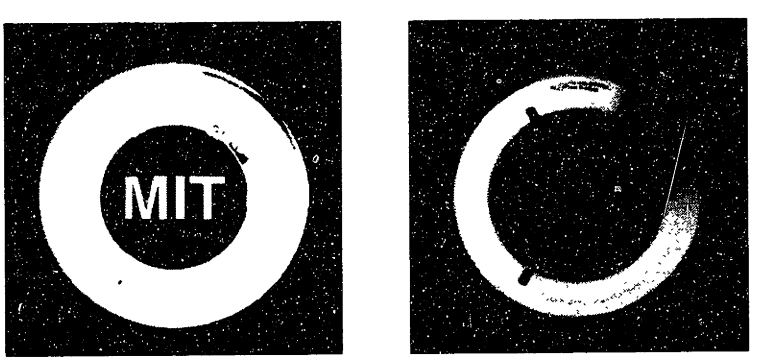

Figure 2.5: Images captured with linear compression at two different iris settings.

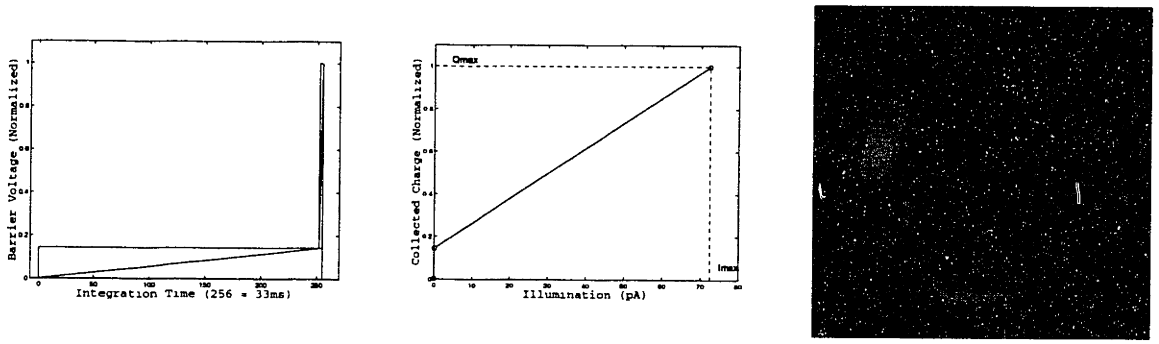

Figure 2.6: Two-step barrier function, Q vs. I plot, and Image data.

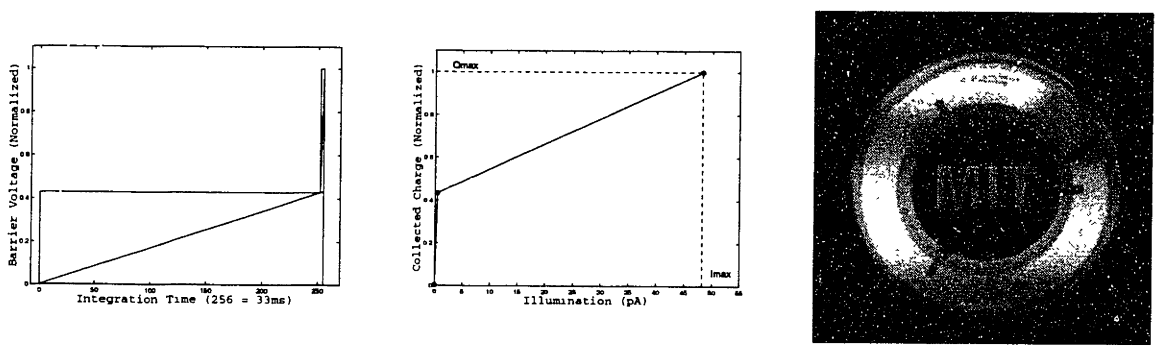

Figure 2.7: Two-step high barrier function, Q vs. I plot, and Image data.

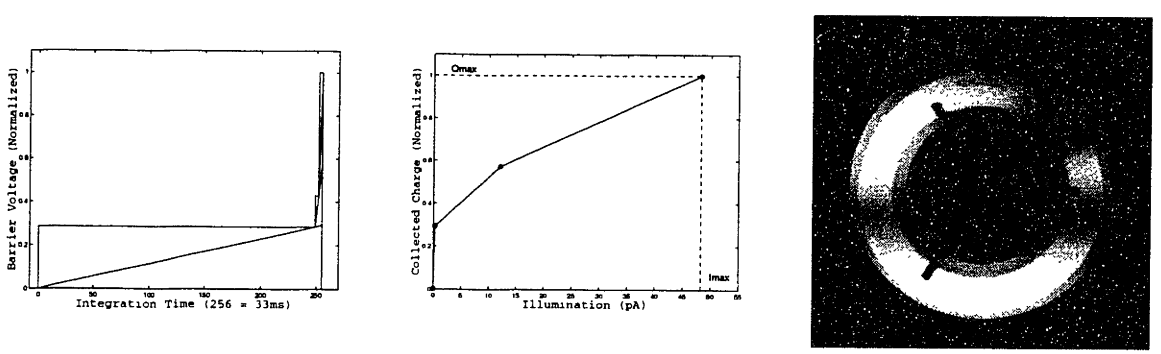

Figure 2.8: Three-step barrier function, Q vs. I plot, and Image data.

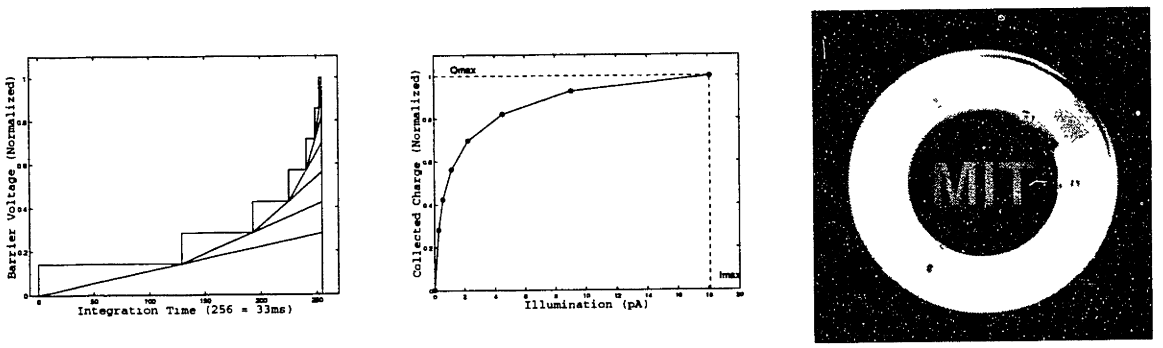

Figure 2.9: Seven-step barrier function, Q vs. I plot, and Image data.

Table 2.1
Summary of imager performance.

Parameter	*Measured*
dark response	174 mV/s @ 22° C
responsivity	23 mA/W @ 550 nm
conversion gain	13.1 μV/e$^-$
FPN (dark)	4.0 mV (1 σ)
power dissipation	52 mW @ 30 Hz, 5 V (10 bits)
saturation level	1.69 V
random noise (dark)	0.56 mV (1 σ)
DR (linear mode)	3000

2.3 Adjusting Global Integration Period

By varying the starting point of the barrier function, the integration period can be adjusted. Figure 2.10 shows the linear and logarithmic compression functions for part of a frame period where the global integration period has been decreased. Varying the starting point of the integration time is similar to varying the shutter speed on a still camera. This is achieved by holding the pixel at the reset value until charge integration is permitted to start. By automatically adjusting the start of the integration period based on the image data, electronic-irising can be employed.

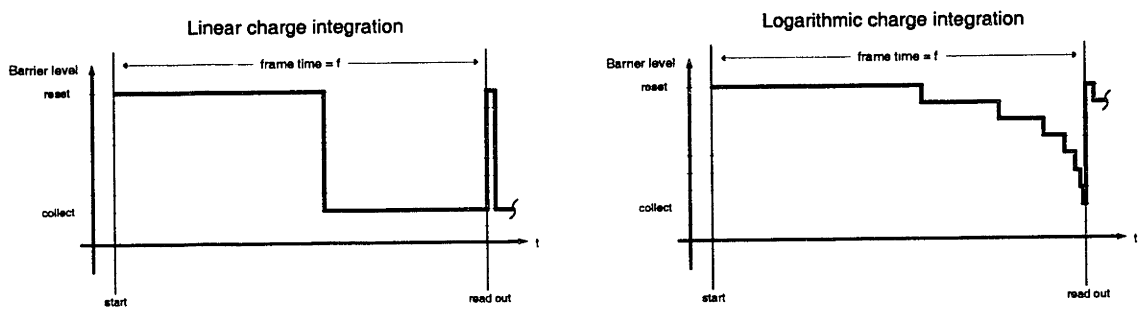

Figure 2.10: Compression functions with decreased integration time.

Furthermore, by adjusting both the global integration time and the voltage steps on the lateral overflow transistor, customized compression schemes can be created for various scenes. Automatic brightness adaptation will therefore require both electronic-irising and automatic compression adjustments. Logic must be produced to control these parameters automatically based on the image data.

2.4 Compression Function Used for Automatic Brightness Adaptation

In order to decide how to adjust the compression function to the image, many iterations must be made in the function fitting process. If the image does not contain a lot of mid-range intensities, the two-step barrier function which maximizes the dynamic range may be used. However, adjusting to such a compression function may result in a large number of invalid frames before a decision can be made. Choosing just two general compression functions (linear and log) will allow good results because only the adjustment of the integration period is necessary after the decision for linear or log compression is made.

Determining the transition points for the steps in the barrier function with varying global integration times may be easily performed if the correct function and barrier levels are chosen. By dividing the remaining interval by 2 after each step, the next step transition point is determined. If the barrier step levels are all equally spaced, the correct function is calculated except for the last transition point. By simply making the last barrier step level twice as big as the previous ones, the correct function is employed. This step can be increased by setting the analog voltage difference for the last two steps to a larger value. Figure 2.11 shows the barrier function used for the automatic brightness adaptive imager in this project. If more dynamic range is desired, or if more emphasis is to be placed on high-illumination, the last step can be made even larger with repect to the previous ones.

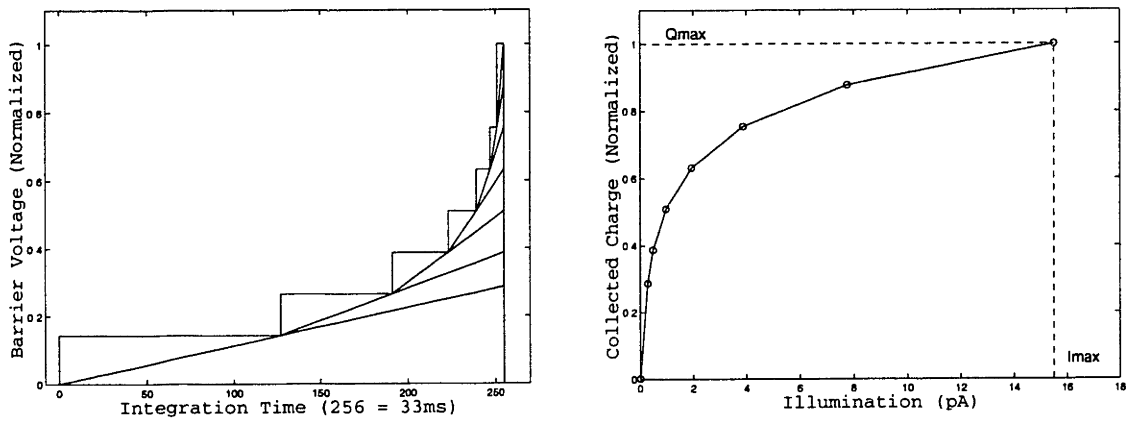

Figure 2.11: Barrier function and compression curve for automatic control.

Chapter 3

Alternative Brightness Adaptive Schemes

This chapter explores other methods used for auto-irising and wide dynamic range imaging. Section 3.1 focuses on specific products and research related to extended dynamic range solid-state image sensors. Section 3.2 focuses on the various techniques used for automatic iris control. Finally a discussion on system-level solutions is given in Section 3.3.

3.1 Image Sensors with Extended Dynamic Range

Dynamic range is defined as the ratio of the largest non-saturating signal in an imager to the random noise of the imager in the dark. Common CCD sensors can acquire an image contrast of about 1:1000 (60 dB). The poor dynamic range is constrained by the dark current of the pixel on the low end and limited by the total amount of charge that can be accumulated per pixel on the high end. Since the dark current decreases with temperature, the dynamic range can be substantially enhanced by cooling the sensor and by employing special readout circuits [15]. However, such methods, in addition to being very expensive, may be inappropriate for real-time applications.

Several approaches for extending the dynamic range have been described and implemented for CCD and CMOS image sensors. The various schemes can be divided into two main types: Sensors utilizing multiple integration periods and sensors employing nonlinear pixel characteristics.

3.1.1 Multiple Integration Periods

Cameras with extended dynamic range have many commercial applications. As a result, some companies are producing high-end imagers with this feature – and patents protecting them. One such company is i-Sight, Inc. Adaptive Sensitivity™ is a proprietary i-Sight technology which enables the acquisition and display of wide dynamic range images [16]. The technology is a visual information processing feature that attempts to mimic the human eye's ability to compensate for uneven illumination in high-contrast scenes. By combining data from multiple images, an optimally contrasted image is created. The Adaptive

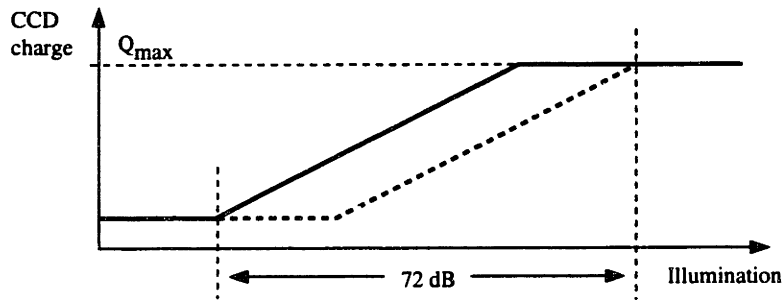

Figure 3.1: Two sample extension of dynamic range.

Sensitivity™ video-processor is a VLSI device for video signal processing in digital cameras.

The algorithm takes advantage of two image exposures. One exposure is made with an electronic shutter set at a short exposure time which captures the brightest details in the image. This causes most of the very dark areas of the image to become lost in the noise. A second exposure of the same image is taken at a relatively long exposure time which contains details of the darker parts of the image. This leaves the brightest areas of the image saturated, without detail. The two images are then combined with the algorithm so as to produce a single, wide dynamic range image. Figure 3.1 shows how the dynamic range is extended by about 12dB.

The Figure 3.2 shows an example of the results produced by the algorithm. The first two images show the areas of a scene requiring a wide dynamic range. Each image lacks the details which may be seen in its counterpart. The last image was produced by the Adaptive Sensitivity™ algorithm and shows the benefits offered by both exposures.

Figure 3.2: Sample photos using Adaptive Sensitivity.

Figure 3.3 is a picture of the iSC2050 camPuter [15], a wide dynamic range digital video camera based on this video processing technology. The camera is capable of obtaining a dynamic range of over 72 dB. The camera has a remote head that is attached to the control unit. The electronic shutter is user selectable from 1/60 to 1/30,000 of a second. The power requirement for the system is 12 volts with a supply current of 1.5A.

Another technique for extending the dynamic range of an imager using multiple exposure periods is explained and implemented by Yang [17]. A 640×512 CMOS imager with floating point pixel-level ADC has been fabricated in a $0.35 \mu m$ process. Figure 3.4 shows the results

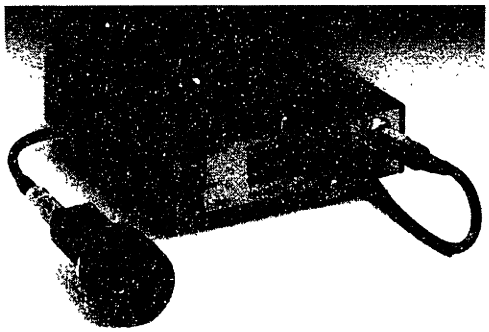

Figure 3.3: The isc2050 Camputer.

of using 8 samples (4 shown) to obtain a wide dynamic range image. The technique involves a pixel-level ADC that is suited to multiple sampling. The expansion in dynamic range has been shown to be 2^k, where k is the number of image samples taken. The combined image has a floating point resolution with exponent k.

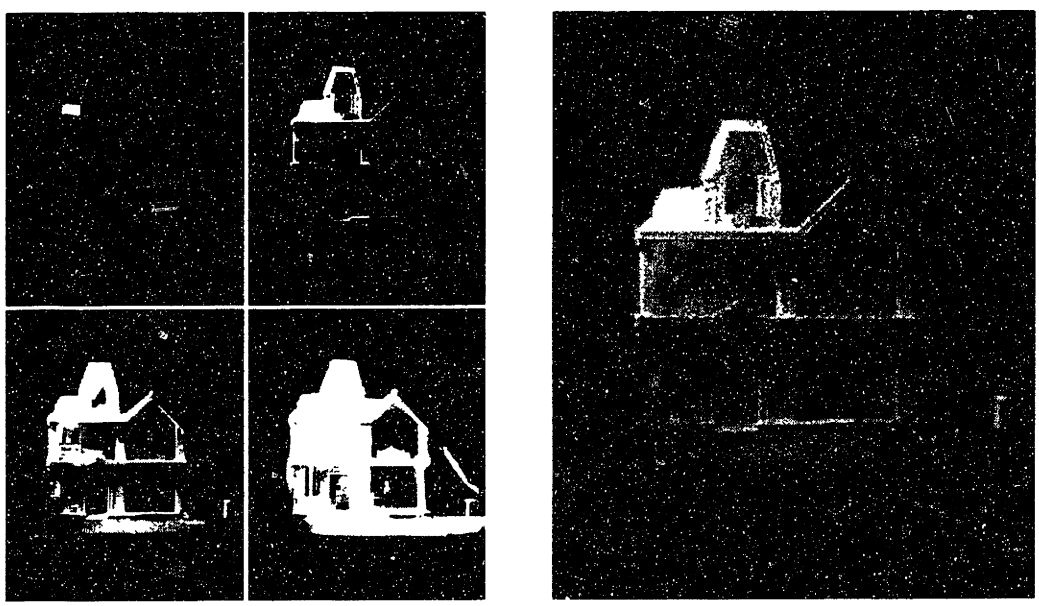

Figure 3.4: Sample photos using floating point pixel ADC.

Researchers at Toyota have been developing wide dynamic range imagers and algorithms for use in autonomous vehicles, as described in [18], [19], [20]. One of the most recent sensors has a dynamic range of 10,000 which combines images taken under different exposure conditions. The effectiveness of the developed vision sensor in comparison with a conventional video camera was confirmed from experiments on a highway under various lighting conditions.

3.1.2 Non-Linear Pixels

Delbruck [21] describes a photoreceptor circuit that can be used in massively parallel analog VLSI silicon chips to perform initial analog visual information processing. The receptor provides a continuous-time output that has low gain for static signals, and high gain for transient signals that are centered around the adaptation point. The response is logarithmic, which allows for a dynamic range of more than 6 decades. The 5-transistor receptor uses an adaptive element that is resistant to excess minority carrier diffusion. The continuous and logarithmic transduction process makes the bandwidth scale with intensity.

Rowley [22] has developed a continuous-time, logarithmic photoreceptor which exhibits an improvement in the signal-to-offset ratio at low and medium light intensities. The logarithmic receptor gives a larger dynamic range than many other continuous-time receptors. The research has shown that the offsets in the photo-conversion elements decrease as the pixel current levels increase.

A floating-gate photosensor used for detecting wide-band photosignals that has high dynamic range and low noise has been developed by Kub and Lin [23]. Low noise is achieved by converting photocurrent to a voltage through capacitive coupling rather than the using a load resistor. The floating-gate photosensor has demonstrated low noise and a dynamic range of 10,000:1. The detector is AC coupled and has a square-root compressing transfer function for the highest light intensities.

Vietze [24] has introduced a pixel structure that exhibits an enhanced dynamic range by subtracting an offset current from the pixel while retaining a linear readout characteristic. The offset current can be programmed by a specific programming voltage. This circuit may be suitable for conventional photodiode sensor architectures as well as for active pixel sensor (APS) designs. Experimental results have shown an increase in dynamic range.

NEC Corp [25] has developed technology for narrow-channel effect suppression in photodiodes for reduction of smear in order to improve dynamic range in small pixel interline-transfer CCD image sensors. The new technologies have been applied to progressive-scan CCDs which show improvement in the pixel charge capacity and dynamic range.

Hamamoto [26] proposes a motion adaptive sensor for image enhancement and wide dynamic range imaging. The motion adaptive sensor is able to control integration time pixel by pixel. The integration time is determined by saturation and temporal changes of incident light. The idea is to obtain high temporal resolution in the moving area and high SNR in the static area.

Bohm et al. [27] have developed image sensors in a TFA (Thin Film on ASIC) technology. A TFA prototype for automotive vision systems has been presented which allows each pixel to adapt its individual sensitivity to the local illumination. A dynamic range of greater than 120 dB has been reported.

3.2 Techniques for Automatic Iris control

3.2.1 Mechanical Irising

There are two common types of automatic iris lenses. The first type, called the *Video Auto Iris*, uses the video signal as feedback to the iris control motor. The other common variant

is called the *DC Auto Iris* which relies on circuitry contained within suitably-equipped cameras. The circuitry monitors the image data and produces a DC output representing the desired iris opening size.

A popular method for iris control in camcorders is described by Furlong [28]. A Hall-effect device is mechanically connected to the iris motor which monitors the iris opening and provides feedback for the opening size of the diaphragm. The voltage output of the Hall-effect switch is typically in the range from 3 volts down to 1 volt. A microprocessor inside the camera creates a pulse width modulated (PWM) iris drive signal. The signal is fed to the iris motor via the drive amplifier. The duty cycle of the PWM signal determines the size of the iris opening. A closed-loop feedback path allows the iris opening to accurately track the PWM drive signal. The microcontroller then selects the iris opening size based on the incoming video signal.

3.2.2 Electronic Irising

Auto iris control that does not involve mechanical parts is usually referred to as electronic irising. There are two basic methods used for electronic irising: automatic shutter speed control and automatic gain control.

The most common fixed shutter speeds for video cameras are 1/60, 1/120, 1/180, 1/250, 1/500, 1/1000, 1/4000, and 1/10,000th of a second. The clocking schemes in many CCDs have been adapted for electronic irising. Since the signals from a CCD pixel array are usually analog, the control unit for setting the shutter speed is also analog. Proper feedback and filter design can lead to very quick response times. In general, electronic irising provides a much quicker response than a mechanical auto-iris.

Automatic gain control can be implemented in the "front end" of a CCD to control the voltage range at which the pixel level registers its data. This technique is not very useful if a wide dynamic range image is desired. Since the dynamic range is limited by the dark current in the low end and the pixel's charge capacity in the high end, using gain to adjust the signal level only reduces the range in either direction.

3.3 Systematic Solutions for Intelligent Vehicles

The Matsushita Audio and Video Research Lab have developed a technique for automatically adapting to a wide dynamic range image [29]. A variable and nonlinear gamma characteristic is applied to the input image depending on the distribution of the luminance. The gamma characteristic is decided so as to amplify the luminance of the dark pixels and to preserve the contrast of the bright pixels for the back-lit objects. Apparently, the output luminance is set to the input luminance for the front-lit objects. A decision rule for the gamma characteristic has been established using the learning algorithms of neural networks. The idea is to make the decision rule coincide with human vision decision rules. The effect of the method is expansion of the dynamic range by about 10 dB. A cascaded connection of RAMs has been developed for the implementation of the gamma decision rule. The adaptive gamma processing has been designed into a consumer video camera.

Systematic solutions for autonomous control are image sensors that adapt to the environment and interface to high-level processors. Although many solid-state imagers now

employ electronic irising, very few companies address solutions for systems. Many of the algorithms described in Section 3.1.1 use off-the-shelf CCD arrays. Actively adjusting an imager to its lighting conditions requires a custom pixel structure designed for a specific set of algorithms. The right combination may require work in both areas simultaneously.

Chapter 4

Design

4.1 Overview

In order to accomplish all of the design objectives of this project it was necessary to build the imager system in stages. The first part of the project involved demonstrating the imager in real-time on a standard monitor with linear and logarithmic compression functions. The next part of the project involved designing a digital imager with a twisted pair interface that could be synchronized to other imagers. The imager would receive and decode any compression function or iris value. The third stage of the project involved the design and evalution of the automatic brightness adaptive algorithm. This included the hardware for electronic-irising and automatic compression. Finally a 3-camera system was built based on the experience and results of the preceding stages of the project.

4.1.1 Real-Time Demonstration

This part of the project involved generating all of the timing signals for the imager as well as the discrete circuitry for the analog sources. The timing circuits are used for controlling the integration period as well as selecting the barrier function. The imager also contains switched-capacitor ADC and CDS circuitry for which the timing signals must be generated. A surface mount PC board was made to keep the size of the imager to a minimum. An NTSC encoder was incorporated into the programmable logic as well as a feature for switching between linear and logarithmic compression functions on the fly. Two discrete blocks of SRAM are used to store an incoming frame and to display the previous frame. The CPLD then ping-pongs between the two memory blocks, depending on the cycle. A D/A converter is used to convert the digital data to an analog NTSC signal for display on a standard monitor. Figure 4.1 shows a block diagram of this system.

4.1.2 Digital Imager

A digital imager was designed to plug into the controller board using a twisted-pair cable connection. A 256 macrocell CPLD was used to provide all the timing for the chip as well as the decoding and encoding of incoming and outgoing data. In order to send data at rates

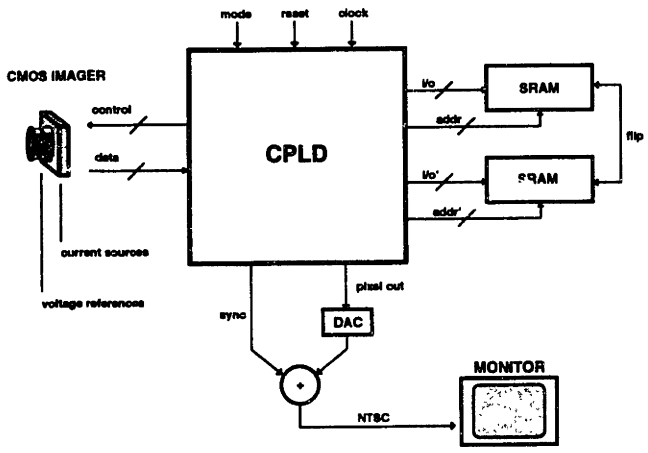

Figure 4.1: Real-time imager with brightness adaptive modes.

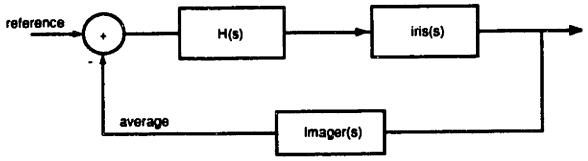

Figure 4.2: Iris feedback controller.

above 50 MHz, low voltage differential signal (LVDS) drivers and receivers were used. The lens cover was designed to accept either CS-mount or C-mount style lenses.

4.1.3 Automatic Brightness Adaption System

The Automatic brightness adaptive imager consists of both electronic irising and intelligent compression. For both features, average frame values are calculated with dedicated hardware as the imager sends data to the host processor. The electronic irising algorithm uses the average of the entire frame and the current iris value in order to calculate the next iris value. The intelligent compression controller uses the averages from 16 subregions in order to approximate the dynamic range of the entire image. An efficient design is presented here that makes use of the column-parallel nature of the imager data.

Electronic Irising

The objective of the electronic iris is to keep the imager's frame average intensity at a desired reference level. A feedback controller is designed to drive the frame average to the reference by using the iris value as the output control signal. A block diagram of the control system is shown in figure 4.2.

The compensator for this feedback controller should allow fast settling time with the lowest overshoot possible. If the effect of image lag is neglected, the only dynamics associated with the imager is the delay from one frame to the next. When an iris value is chosen, the frame average is immediately updated on the next available sampled frame

value. Rather than operating on the difference between the frame average and the reference value, a simple calculation for the iris at every other sample point will prove to give the best results. The iris control variable that sets the integration time can be thought of as a mechanical shutter to the imager. If the illumination in the environment stays constant, then there is a linear relationship between the position of the shutter and the brightness of the image on the surface of the pixel array.

The iris value that generated the frame average at each sample is known. Since the relationship between the iris value and the frame average is also known, the illumination level of the room can be calculated. Once the illumination level is determined, the next iris value can be selected. If the illumination in the room stays constant over the next frame period, the frame average will be driven directly to the reference value in the subsequent frame. The equation for controlling the iris value uses the following parameters:

$$iris \longleftarrow \begin{cases} 255 & \text{when } closed \\ 0 & \text{when } open \end{cases} \tag{4.1}$$

$$frame \longleftarrow \begin{cases} 255 & \text{when } bright \\ 0 & \text{when } dark \end{cases} \tag{4.2}$$

$$ref \longleftarrow 128 \equiv \text{half-scale} \tag{4.3}$$

The equation is then given by:

$$iris_n = \frac{ref}{frame_{n-1}} \times iris_{n-1} \tag{4.4}$$

The relationship between the iris value and the frame average shown in equation 4.4 is used for the electronic-iris in the digital imager. Therefore, controlling the integration time of the imager involves dividing a previous integration time by the frame average and shifting the result. The hardware implementation involves an accumulator and a cyclic divider.

Automatic dynamic range adjustment

The two main issues with changing the compression scheme during the integration period are how to measure the dynamic range of the image and how many different compression functions to use. Figure 4.3 shows one way of estimating the dynamic range of an image by breaking it into smaller blocks and calculating the average intensity of each region. By comparing the average intensities of the individual blocks, a wide dynamic range image can be targeted.

For the purposes of this project, the system will either choose between a linear compression function or a logarithmic compression function. The system will use the 16×16 blocks to estimate the dynamic range. A ratio of the brightest block to the darkest block can give a good estimate of the dynamic range if the imager is also using electronic irising. The estimate of the dynamic range will have different values depending on the compression function that is employed during the calculation. Therefore, the dynamic range estimate is compared against a linear threshold value when the linear compression function is used

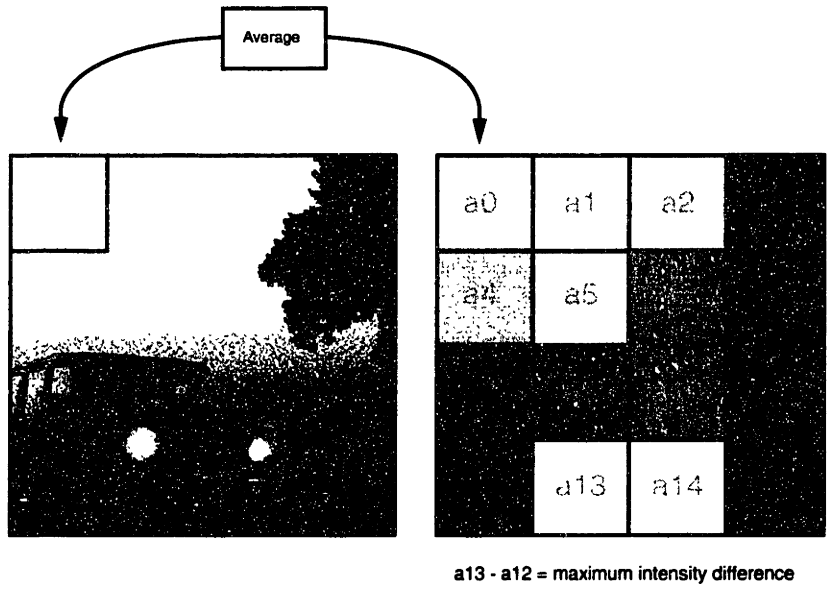

Figure 4.3: Algorithm for measuring a wide dynamic range image.

and against a logarithmic threshold value when the logarithmic compression funtion is used. The decision rule for the compression function is given below:

When in *linear* mode:

$$\begin{aligned} if \text{ dynamic range} &> \text{linear threshold } then \\ \text{mode} &\leftarrow log \end{aligned}$$

When in *log* mode:

$$\begin{aligned} if \text{ dynamic range} &< \text{log threshold } then \\ \text{mode} &\leftarrow linear \end{aligned}$$

The difference between the two threshold values causes a hysteresis in the decision rule which minimizes the possiblity of oscillating between modes in border-line cases.

4.1.4 PCI Interface

In order to get three 256×256 images at 60 fps into the PC a specialized PCI interface board would have to be designed. The software radio group at MIT had already designed a data acquisition board that would meet the design needs for this project [30], [31]. The General Purpose PCI Interface (GuPPI) card was easy to use and already had drivers written for linux. The formatted data was grouped into 32-bit words with the last 24 bits containing 8 bits from each imager. The upper 8 bits were used for the vertical sync and reserved for any extra information that might need to be sent to the PC. The only necessary signals for loading the data into the GuPPI card were the clock and write-enable signals. The GuPPI card also sends 4 control signals to the board that could enable or disable features such as electronic irising and auto-compression.

4.2 Real-Time Camera Implementation

Generating the logic for the various stages of this project involved creating modules in VHDL for programming CPLDs. The following sections describe some of the modules that were used in the first real-time demonstration of the imager. Nearly all of the modules described in this section were used again in later stages of the project.

4.2.1 Format Converter

The format of the imager's pixel data has been described in Chapter 2. The image datapath is broken up into bit-planes by row. In essence, each row is separated into two parts. All of the bits for the first part are sent in bit-planes, starting with the MSB. Upon completion of the transfer of the LSB bit-plane, the second half of the row is sent in the same format starting with the MSB. Each subsequent row is sent in the same manner until the entire frame is finished. In order to reformat the data into a raster scan format, at least half a row must be buffered at a time. However, if the data must be sent in NTSC format, an entire frame must be buffered because the read out operation is faster than the write in operation. This is due to the horizontal and vertical blanking periods of the NTSC standard, as will be described in Section B.1.25. Therefore, the strategy for reformatting the data is to perform part of the reformatting while writing data into SRAM and the final formatting on the read out operation.

A diagram of the output pins of the imager is shown in Figure 4.4. Each of the 32 output pins is responsible for 8 columns located near the lines shown in the figure. The details for the order of the column outputs for one-fourth of the imager is shown in Figure 4.5. The A and B select signals are generated along with the signals that drive the ADC and CDS circuits. Since every two columns share one ADC, the A and B select signals are necessary to control which column to convert. Due to the nature of the ADC, all of the bits for the columns in A must be generated before switching to the columns in B. The data for each row is therefore broken down into two halves that are separated by the periods of the A and B select. For example, the output sequence of pin 16 is listed below with the notation, $column_{bit\_number}$:

$$A : 1_7, 5_7, 9_7, 13_7 \rightarrow 1_6, 5_6, 9_6, 13_6, \ldots, 1_0, 5_0, 9_0, 13_0 \rightarrow$$
$$B : 3_7, 7_7, 11_7, 15_7 \rightarrow 3_6, 7_6, 11_6, 15_6, \ldots, 3_0, 7_0, 11_0, 15_0$$

The object of reformatting the data is to put pixels together starting with the MSB down to the LSB. In order to do this, each 32-bit word that comes off the imager during the output cycle is broken down into eight 4-bit words. Using SRAM that stores 8-bit words allows two 4-bit blocks to be stored at each word location. Table 4.1 shows which imager outputs are mapped to the 4-bit blocks. The idea is that by writing four bits from separate columns in the first half of the word, the second half of the word is reserved for the next four bits from the same columns. This allows any column's MSB to be written to the same word as the $MSB - 1$. The process is then a matter of first reading out the previously

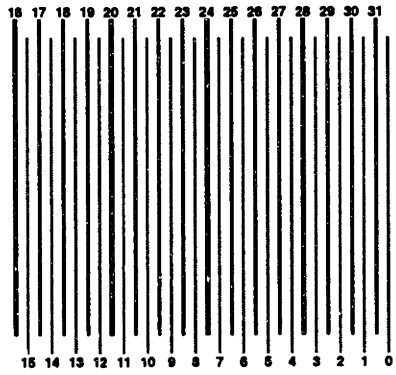

Figure 4.4: Imager output lines.

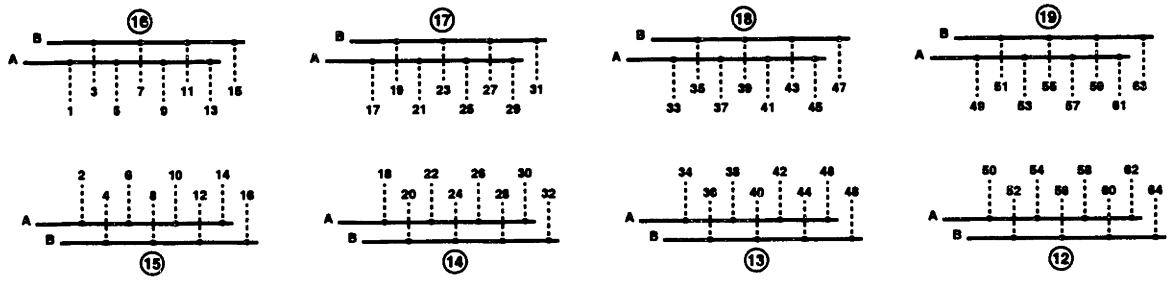

Figure 4.5: Output format for each pin.

written four bits from SRAM before storing the next four bits along with the previous four bits.

The key to formating the data is that the address counter for the SRAM gets shuffled in a particular way for the write in operation and in a different way for the read out operation. Figure 4.6 shows how all the data from the imager gets stored in SRAM.The signals *inc*, *n*, *pac*, *part*, *prime* and *row* all correspond to aliases for the bits on a 17-bit counter. The

Table 4.1
Imager output pins.

4-bit Block	*Imager Output Pins*			
A	16	15	17	14
B	18	13	19	12
C	20	11	21	10
D	22	9	23	8
E	24	7	25	6
F	26	5	27	4
G	28	3	29	2
H	30	1	31	0

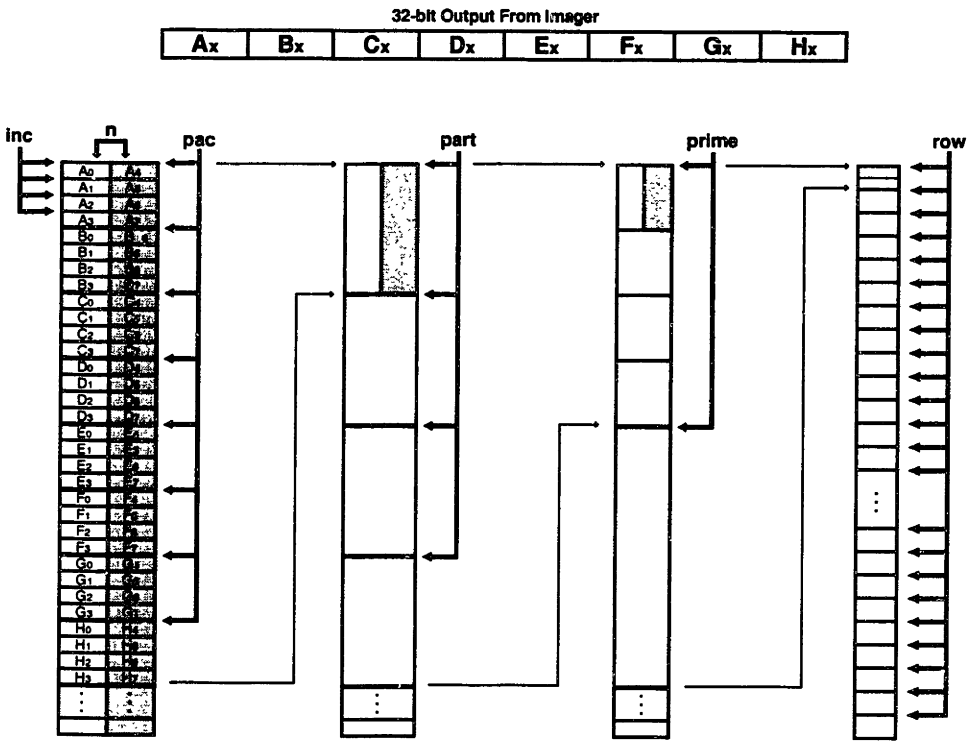

Figure 4.6: Format converter.

counter increments when each 4-bit block from the 32-bit imager data is handled. The aliases have the following associations: *pac* stands for each of the eight 4-bit blocks from the imager data, *inc* is for the four columns associated with the A or B select, n controls the write in or read out operation, *part* is for the four parts of the pixel, *prime* indicates the A or B select cycle, and *row* is the row number.

The aliases are combined together to form the shuffled address for the SRAM. Figure 4.7 shows how the aliases are assigned to the counter and mapped to the address lines for the SRAM. The timing for writing into SRAM is governed by the timing of the imager. A state machine controls the operation of the read and write operations. When data from the imager is ready, the state machine cycles through 8 reads and 8 writes from SRAM. The bottom 4 bits from each word in SRAM are combined with a 4-bit block from the imager and then written back to SRAM. This causes two adjacent bit-planes to be stored in the same word location for the SRAM. An example of the first four words in SRAM is given below:

$$
\begin{array}{cccccccc}
1_7 & 2_7 & 17_7 & 18_7 & 1_6 & 2_6 & 17_6 & 18_6 \\
5_7 & 6_7 & 21_7 & 22_7 & 5_6 & 6_6 & 21_6 & 22_6 \\
9_7 & 10_7 & 25_7 & 26_7 & 9_6 & 10_6 & 25_6 & 26_6 \\
13_7 & 14_7 & 29_7 & 30_7 & 13_6 & 14_6 & 29_6 & 30_6
\end{array}
$$

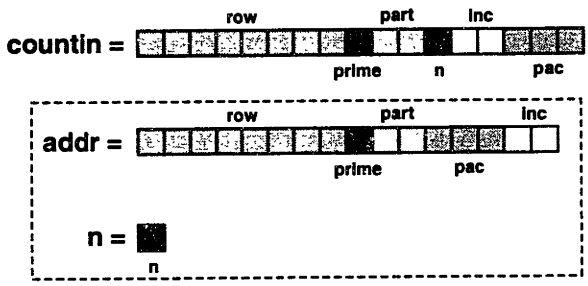

Figure 4.7: Mapping of input counter to SRAM address.

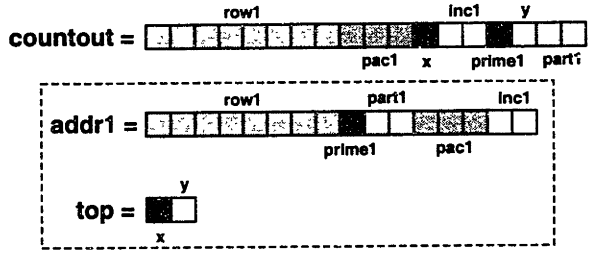

Figure 4.8: Mapping of output counter to SRAM address.

The read out cycle is governed by the timing for the NTSC encoder. The encoder gives a signal to start a new frame, followed by a row start signal. In order to turn out one 8-bit pixel, it takes 4 reads from SRAM. An 18-bit counter is used as the pixel counter and new aliases are shuffled and mapped to the address bits for SRAM. Figure 4.8 shows the aliasing for the output counter and the mapping to the SRAM bits. The aliases $inc1$, $n1$, $pac1$, $part1$, $prime1$ and $row1$ have the same significance as the aliases from the input counter. The variable *top* is used for masking the words that are read off of the SRAM. Another variable called *bot* is simply $top - 4$ and together they select only the top and bottom bits for the same column of an 8-bit word from the SRAM. After 4 reads from the SRAM, a variable called *pixel* becomes full and is sent to the output as a correctly formatted word. Figure 4.9 gives a summary of the operation of the write in and read out cycles.

The CPLD controls the swapping of the I/O ports and the address lines for the two SRAM chips. A variable called *flip* is used to ping back and forth between SRAM chips. The VHDL code for the format converter is included in Appendix A under the file name inout.vhd. The updated format converter used in the automatic brightness adaptive system is found in Appendix B.

4.2.2 NTSC Encoder

In order to display the images in real-time on a monitor, an NSTC encoder was included in the design. Off-the-shelf NTSC encoders were not well-suited to this project because the packaging of the encoders was too large to fit in the desired PCB size. Also, running an NTSC encoder meant that a complicated network of handshaking and clock division would need to be implemented. Instead, an NTSC encoder was written in VHDL and programmed

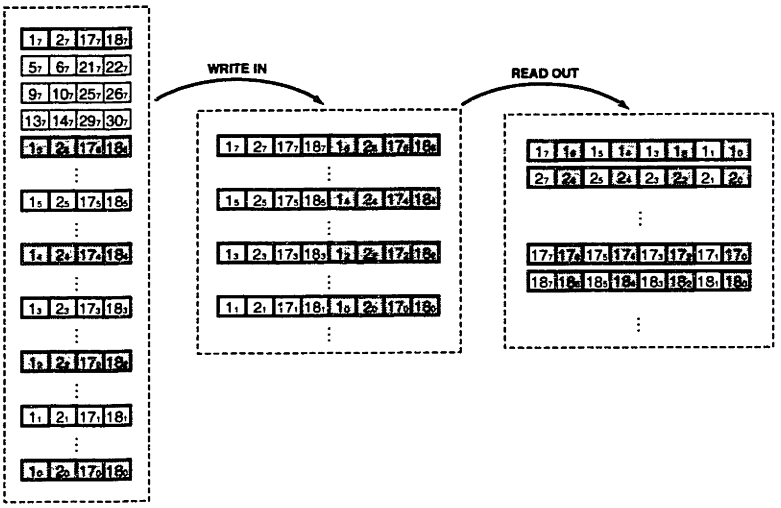

Figure 4.9: Summary of format converter.

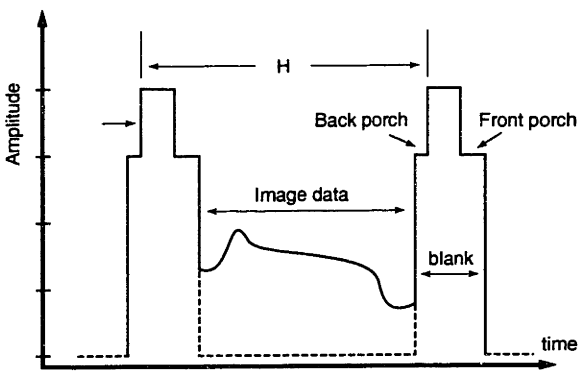

Figure 4.10: Details of horizontal blanking and sync pulses.

into the same chip that generated the timing signals for the imager. The only signals that needed to be generated for the encoder were a *sync* signal and a *blank* signal. The *sync* signal is a combination of vertical and horizontal syncs. The *blank* signal is used to blank out the electron beam of the picture tube during vertical and horizontal retrace. Figure 4.10 show the characteristics of the horizontal sync and blanking periods. The vertical sync is made up of *equalization* and *seration* pulses. A listing of standard NTSC pulses is included in Table 4.2.

Since it takes 4 clock cycles to get out one pixel from SRAM, the pixel clock was set to the system clock divided by 4. The ratio for visible line time to the total horizontal line time is 0.84. Since the imager has a square array of 256×256 and a monitor has an aspect ratio of 4×3, a small section on each side of the monitor is blanked out. The actual visible line time, then becomes $0.84 \times 3/4 = 0.63$. Since it takes 1024 clocks to show all 256 pixels, the total number of clock cycles for the entire horizontal line should be $1024 \times 1/0.63 \approx 1625$. The system clock was selected to be $24.576MHz$ because it can be divided down to exactly $60Hz$.

Table 4.2
Details of NTSC signals.

PERIOD	TIME
Total line (H)	$63.5\mu s$
H blanking	$0.14H = 9.5 - 11.5\mu s$
H sync pulse	$0.08H = 4.75 \pm 0.5\mu s$
Front Porch	$0.02H = 1.27\mu s$
Back Porch	$0.06H = 3.81\mu s$
Visible line time	$0.84H = 52 - 54\mu s$
Total field (V)	$262.5H$ or $1/60 = 0.0167s$
V blanking	$0.05V - 0.08V$
Each V sync pulse	$27.35\mu s$
Total of six V sync pulses	$3H = 190.5\mu s$
Each E pulse	$0.04H = 2.54\mu s$
Each seration	$0.07H = 4.4\mu s$
Visible field time	$0.92V - 0.95V$

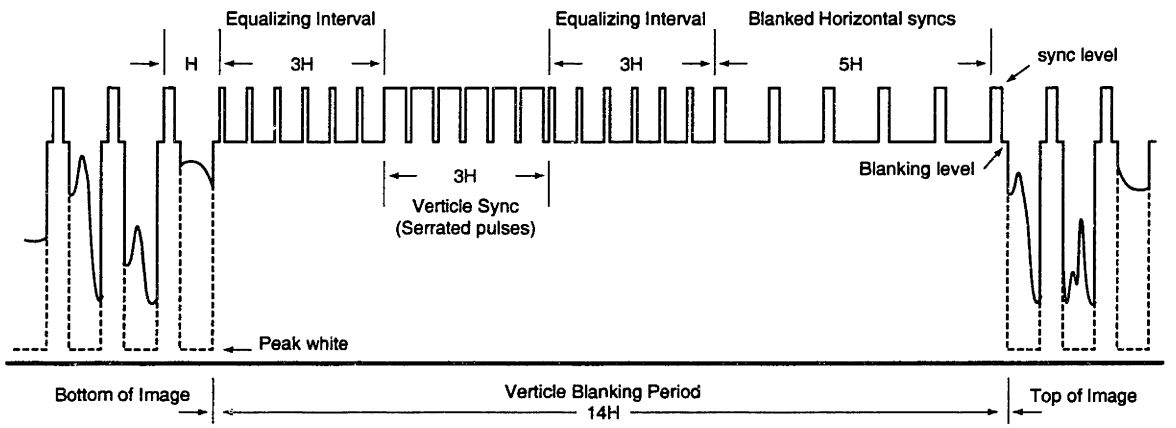

Figure 4.11: Details for the successive fields in NTSC.

The number of clocks per frame is therefore $24.576MHz/60Hz = 409600$. Since NTSC has 262.5 lines per frame, the number of clocks per row should be $409600/262.5 = 1560.3809$. Instead, 256 lines per frame is chosen to give exactly 1600 clocks per row. This works out very nicely because 1600 clock cycles gives the correct aspect ratio on the monitor and allows a consistent row time throughout the image. Since there are fewer lines per frame, the number of blanked lines after the vertical sync is less than an ordinary NTSC signal. This simply reduces the size of the blank region at the top of the image.

Figure 4.11 gives the details of the vertical blanking period during vertical scanning. The vertical flyback starts with the leading edge of the third seration which means that one horizontal line time passes during the vertical sync before flyback starts. Also, six equalizing pulses equal to three lines occur before vertical sync. So $1 + 3 = 4$ lines are blanked at the

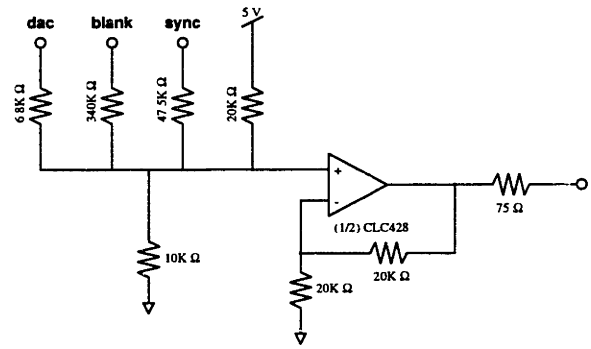

Figure 4.12: Opamp for driving the video monitor.

bottom of the picture just before vertical retrace starts. The amount of time needed for vertical retrace is usually 5 lines. As the scanning beam retraces from the bottom of the monitor to the top, five complete horizontal lines are produced. This leaves 4 lines blanked at the bottom before flyback and 5 lines blanked during flyback. So the total number of blank lines must be at least 9. An ordinary NTSC signal uses 20 blank lines during the vertical blanking period. For this project, 14 blank lines were chosen. This leaves a viewing area with a spatial resolution of 256×242. By slightly altering the vertical blanking period, a very efficient encoder was constructed. The result was 30 frames of data per second with 60 fields per second displayed in non-interlaced format. The VHDL code for the encoder can be found in Appendix B. An interlaced encoder was also developed and included in Appendix A.

A D/A converter was used for generating the analog pixel values for NTSC. When the blank signal is activated, the D/A converter outputs its lowest level. In order to combine the output of the D/A converter and the blanking and sync signals, an opamp was used with a summing junction as shown in Figure 4.12. A 75 ohm resister was used for matching the impedance of the coaxial cable and video monitor.

4.2.3 Imager Timing Signals

The signals that must be generated for the pixel array are listed in Table 4.3 with the digital signals in the upper section and the analog signals in the lower section. The schematics for the analog components can be found in Appendix A. The schematics for the updated analog components of the digital imager is found in Appendix B. The bias voltages $VREFP$, $VREFM$, VCM, $OFFSET$, and $CBIAS$ were generated with fixed resistors acting as voltage dividers with tantalum capacitors at each node. The current sources $oa1bias$, $oa2bias$, and $cbias$ were also fixed resistors from the analog supply to the input pins. The $colx$ voltages were generated from a resistor array with ceramic capacitors on each node.

The digital signals are generated from a CPLD that was programmed from VHDL files. The timing diagram in Figure 4.13 shows the digital waveforms for these signals. The outputs of the imager are 50 times slower than the system clock so a process is run that updates the imager signals every 50 clock cycles. The generation of the $phix$ signals requires some internal states to avoid overlapping edges. This was important because the operation

Table 4.3
Imager pin description.

PINS	DESCRIPTION
rsin	bit for start of frame
swsel1,swsel2,swsel3	selection for output mux
colsel1,colsel2,colsel3	selection of barrier level
clkBB,clkA	latches ADC outputs into mux
phi1,phi2,phi3, phi4,phi5,phi6	clocks for ADC circuit
phi0,phi0bar	signals for start of ADC
Aselect,Bselect	selection for columns to ADC
isolate	hold values on the CDS circuits
sample1,sample2	samples pixel value and reset (CDS)
clkr	advances row select
clkb	advances barrier
out(1 - 32)	the 32 pixel outputs
vcm	common mode voltage for opamp
vrefm,vrefp	voltage references for ADC
offset	offset for CDS
oa2bias	sets bias for ADC opamps
oa1bias	sets bias for CDS opamps
cbias	sets bias for pixel source followers
cgbias	voltage at charge spill device
colsel1,colsel2,colsel3, colsel4,colsel5,colsel6, colsel7,colsel8	voltage levels for barrier function

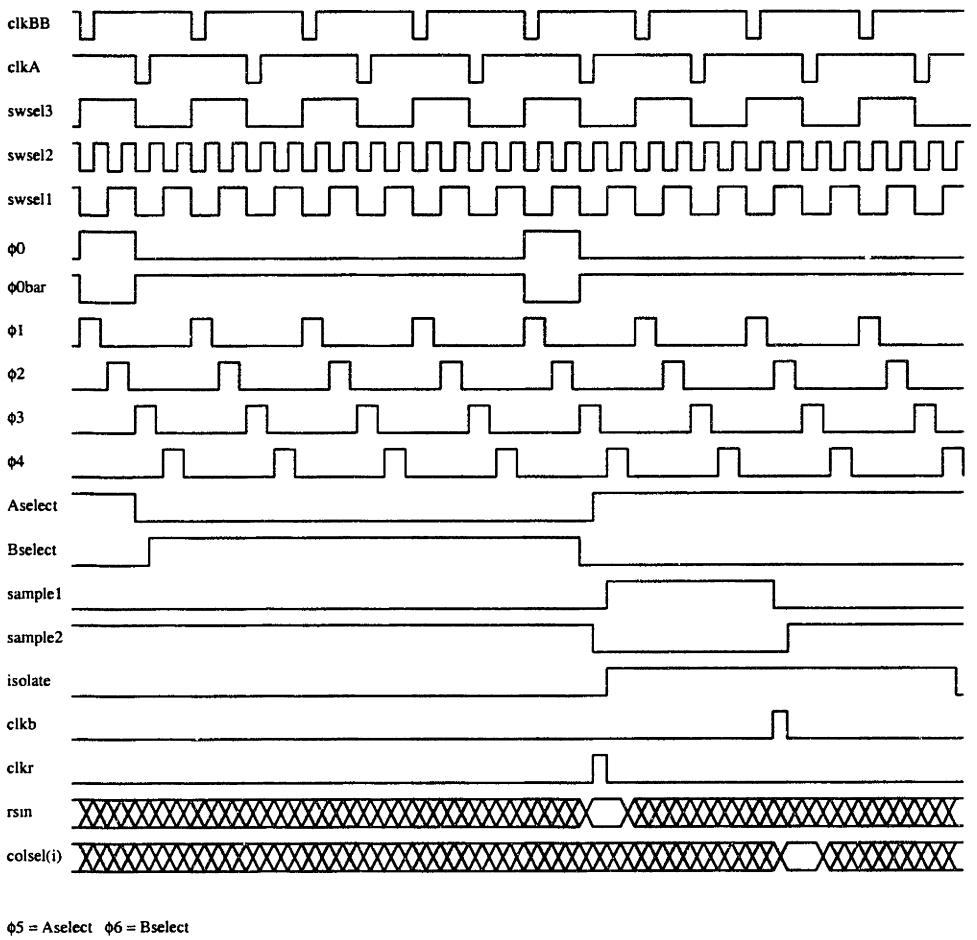

Figure 4.13: Imager timing signals.

of the ADC requires clean, non-overlapping signals.

A choice between 2 barrier functions is selected by an on-board switch. In linear mode. The colsel signals choose only between The highest potential (reset) and the lowest potential($\approx 2.5V$). In compression mode, the 3-bit colsel word increments at the specific row intervals corresponding tc the desired function.

4.3 Digital Imager Implementation

The operation of the digital imager is essentially the same as in the real-time demo except that a new data protocol was developed to send and receive data to and from the host. In order to keep a small link between the imager and host, the data from the imager is sent serially with a clock. The clock line also contains information about the start of each sample and the start of the frame. The imager receives a system clock signal and an input line from the host. The data rate for the link is about 50Mbs, so LVDS drivers and receivers are used with twisted pair media. Figure 4.14 shows the interface between the imager and

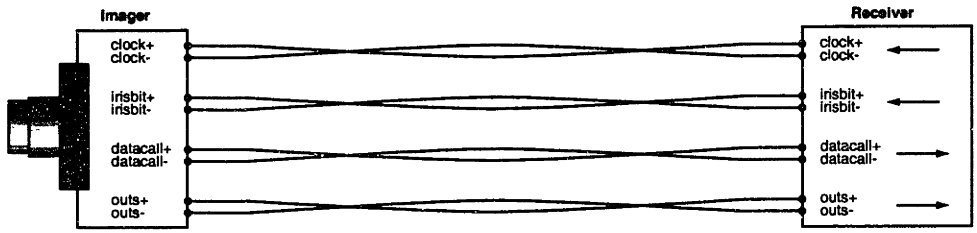

Figure 4.14: Twisted pair interface.

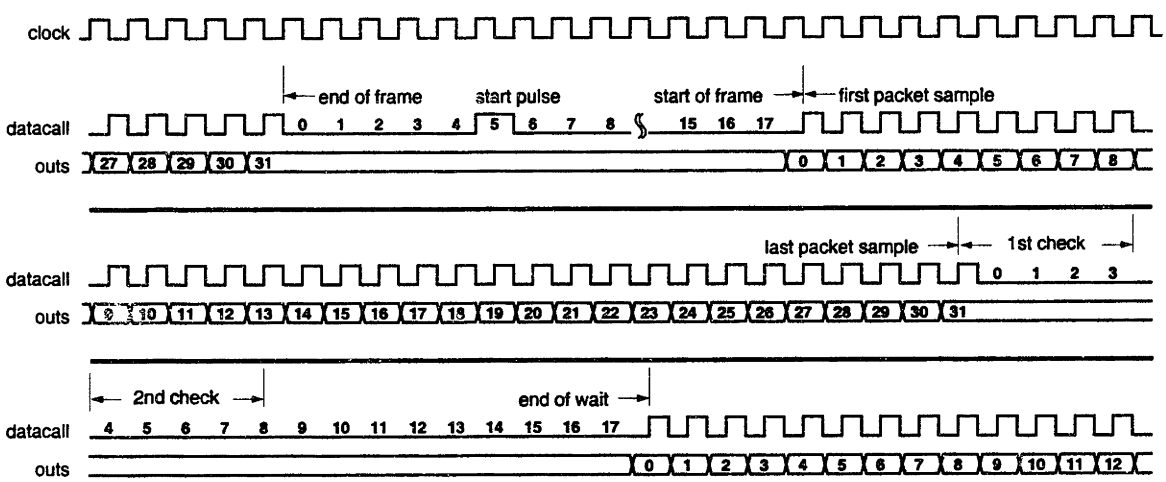

Figure 4.15: Timing for *datacall* and *outs*.

receiver.

The four signals *outs*, *datacall*, *clock*, and *irisbit* are the only signals used to communicate between the imager and the controller. Figure 4.15 is a timing diagram of the signals *outs* and *datacall*. The image data is sent to the receiver in 32-bit packets from *outs*. The signal *datacall* is used to clock in the data at the receiver. A watch dog monitors the wait interval between samples in order to determine if the next packet is at the start of the frame. A wait interval is targeted when four clock cycles pass before *datacall* increments the count register at the receiver. If the second wait interval passes with out a change in *datacall* the count register resets and waits for the next rising edge of *datacall*. However, if *datacall* changes during the second interval, the next packet will be the start of the frame.

The signal *clock* is sent from the receiver and becomes the imager's system clock. This allows multiple imager's to use the same system clock so the all the data is synchronized. Figure 4.16 shows a timing diagram for the transfer of the iris and mode values. The signal *irisbit* remains low until the transfer is about to proceed. It is then sent high for one clock period and low for the next clock period. The transfer then begins with the MSB of the iris value. The last bit that gets transfered is the mode value which indicates the compression function to apply. The signal *irisbit* is also used to reset the imager. When *irisbit* is held high for 16 clock cycles the imager enters a reset state. The imager then starts at the

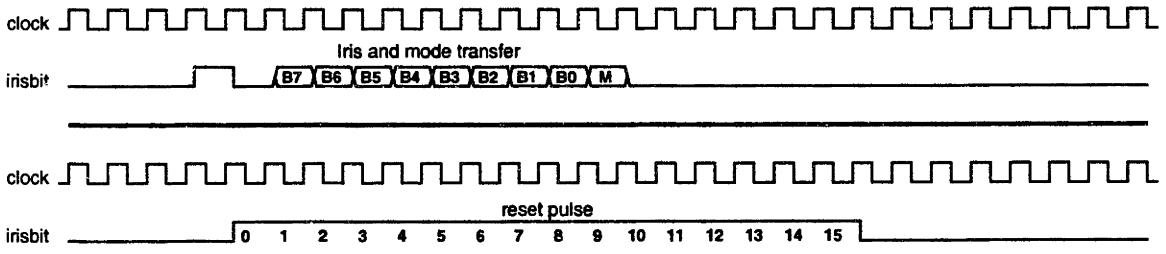

Figure 4.16: Timing for *clock* and *irisbit*.

beginning of the frame when *irisbit* falls low.

4.4 Automatic Brightness Adaptation Controller Board

The automatic brightness adaptive system consists of both electronic irising and automatic mode switching. A test board was made to demonstrate the effectiveness of the algorithm working with just one imager. A block diagram of the system is shown in Figure 4.17. The digital imager plugs into the board with a twisted pair interface. The two serial lines *irisbit* and *outs* are used to send and receive data from the controller board. The data received by the board is formatted and sent to either an NTSC monitor or a standard PC running linux. The shaded feedback path in the figure involves both controlling the charge integration time as well as the pixel compression function. An average of each 64×64 block is computed during the frame time and then used to calculate the dynamic range and the total frame average. These values are then used to set the compression function and the global charge integration time.

The logic required for the electronic-iris and the automatic mode switching consists of two CPLDs shown in figure 4.18. The CPLD *TOPBOX* controls and accumulates the incoming data from the imager in order to calculate the average values of the 16 regions used for the dynamic range calculation. The first block *SERIAL2PARALLEL* converts the serial data from the imager to parallel data using the signal *datacall* from the imager as its shift register clock. A *start-of-frame* signal is also decoded from *datacall* by *SERIAL2PARALLEL*. The parallel data gets sampled and sychronized to the system by *SAMPLE* and is passed on to *CHIPMUX* where the data gets broken into 4-bit blocks. These 4-bit blocks are used by the 4 accumulators contained in *BOXAVERAGE*.

Figure 4.19 shows how the 4-bit data is piped through the accumulators. Each of the accumulators represents one of the 4 columns of the 4×4 block averages used for the dynamic range calculation. The signal *column* is used to select which column average is active in subsequent blocks. The signals *newbitplane*, *newpixel*, and *newblock* are control signals from *BOXCONTROL* which is a finite state machine that is used to control the arithmetic units of the system. The accumulators are essentially sychronous loadable counters that each use one of the 4-bit inputs as the load signal. Therefore calculating the average is only a matter of incrementing a counter. Since the bits from the imager come off in bit-planes ranging from the MSB down to the LSB, the signals *newbitplane*, *newpixel*, and *newblock*

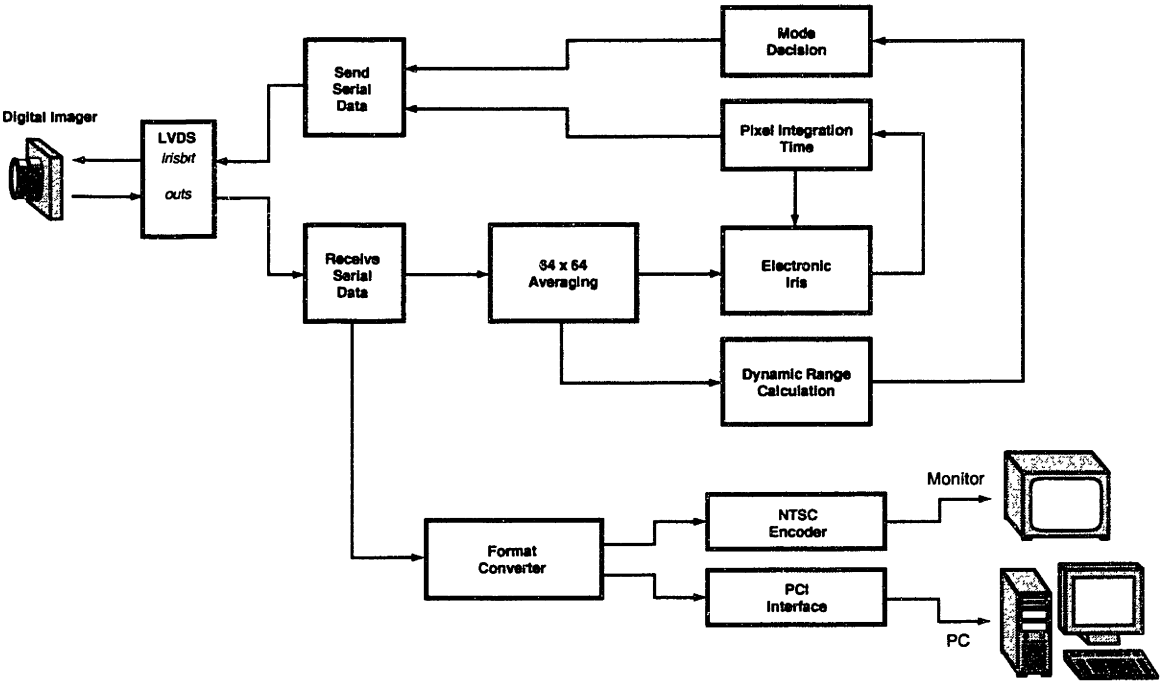

Figure 4.17: Automatic brightness adaptive test board.

are used to shift and load the bits of the counter depending on which bit-plane is being processed. The saturation blocks are used to ensure that an overflow is represented by the highest average value (all ones).

The CPLD called *TOPWITHLIGHTS* contains most of the arithmetic functions for the system. *TOPBOX* sends the control signals and the averages for the 16 regions which are selected by the signal *column* and by the row time during the frame period. The block called *COMPARE* is used to latch in the brightest and darkest region averages for the image. Figure 4.20 shows a logic diagram of the components used to perform the operation. Another accumulator is included in *COMPARE* that adds up the 16 regions so that a total frame average value can be used for electronic irising. The block *DIVIDER* is used to divide the current iris value by the frame average. The division is carried out to 7-bits beyond the decimal which also performs the multiply-by-128 operation described in Section 4.1.3. The divider implemented for this algorithm requires one cycle for every bit output as described by the VHDL files in the Appendix B starting on page 119.

The block *MODEGEN* either sets the compression function to linear or logarithmic. The dynamic range value represented as the ratio of the brightest and darkest regions is compared against the thresholds set by the decision rule described in Section 4.1.3. The *mode-bit* is then combined with the iris value and sent to the imager by the block *IRISGEN*.

The remaining features in the CPLD *TOPWITHLIGHTS* include input pins for switches and output pins for LED indicators. The switches are used to turn on and off automatic irising and compression. The frame average, iris value and dynamic range are also displayed by rows of LEDS. A 4×4 grid of dual-color leds is used to show the brightest

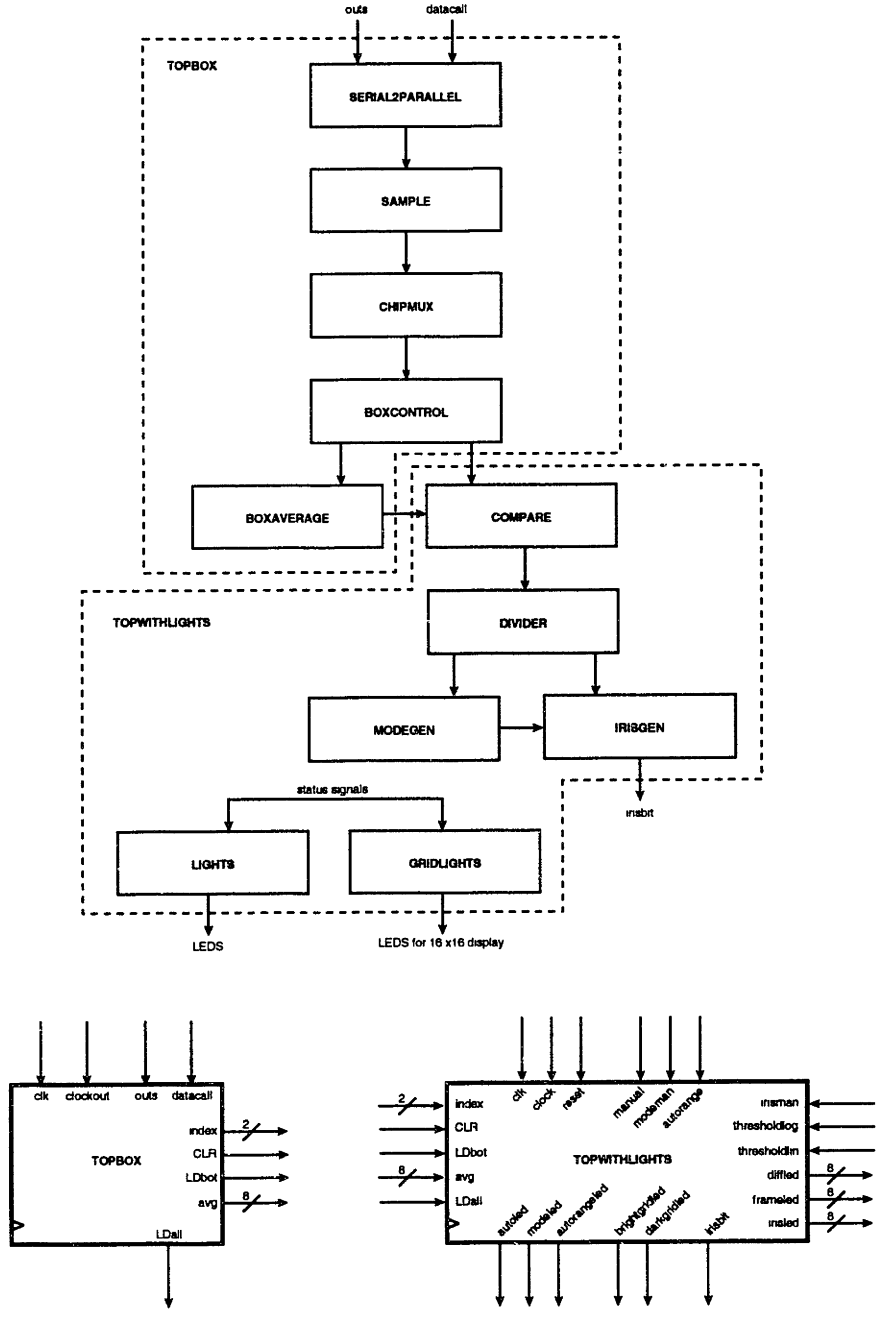

Figure 4.18: Block diagram and symbols of CPLDs for brightness adaptation.

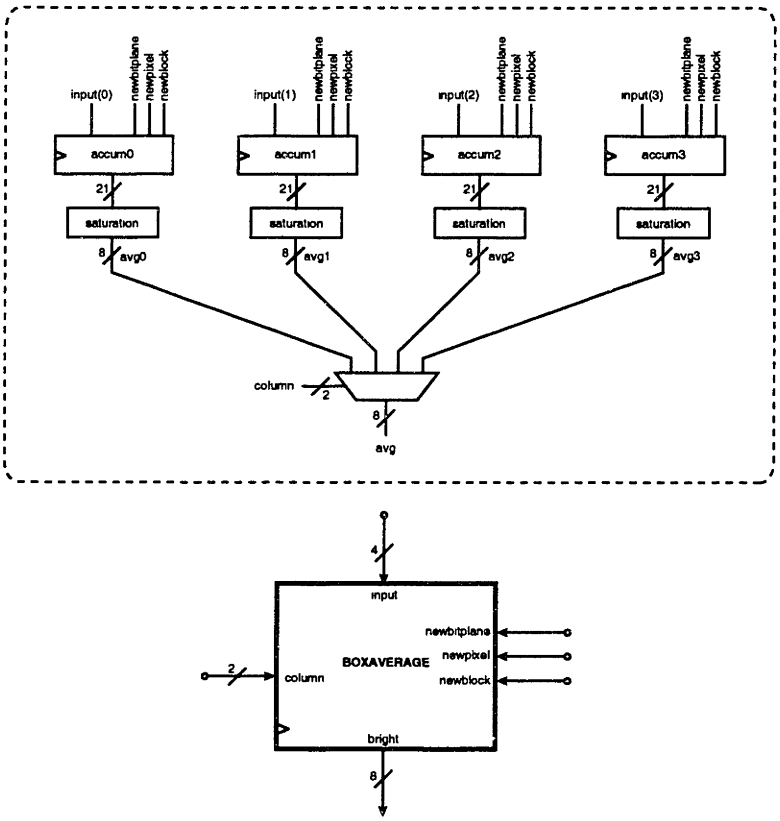

Figure 4.19: Accumulators used for frame averageing.

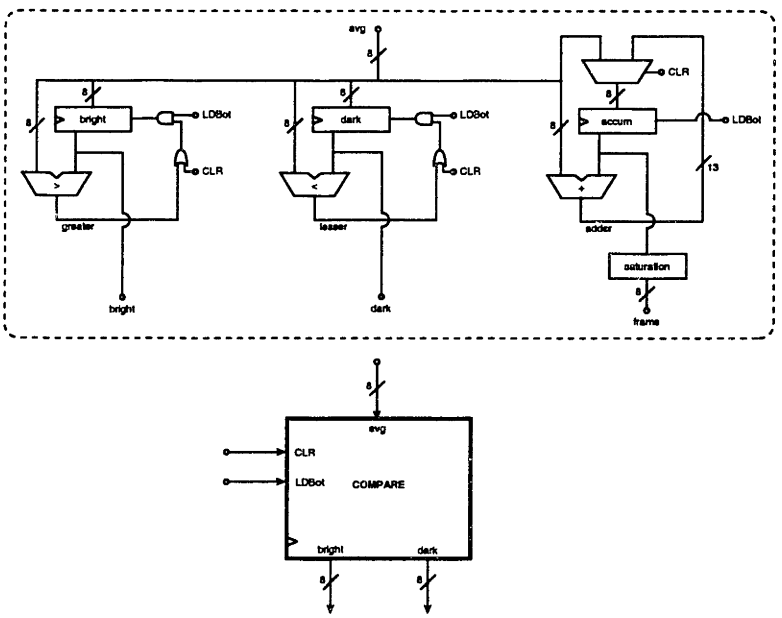

Figure 4.20: Timing for *clock* and *irisbit*.

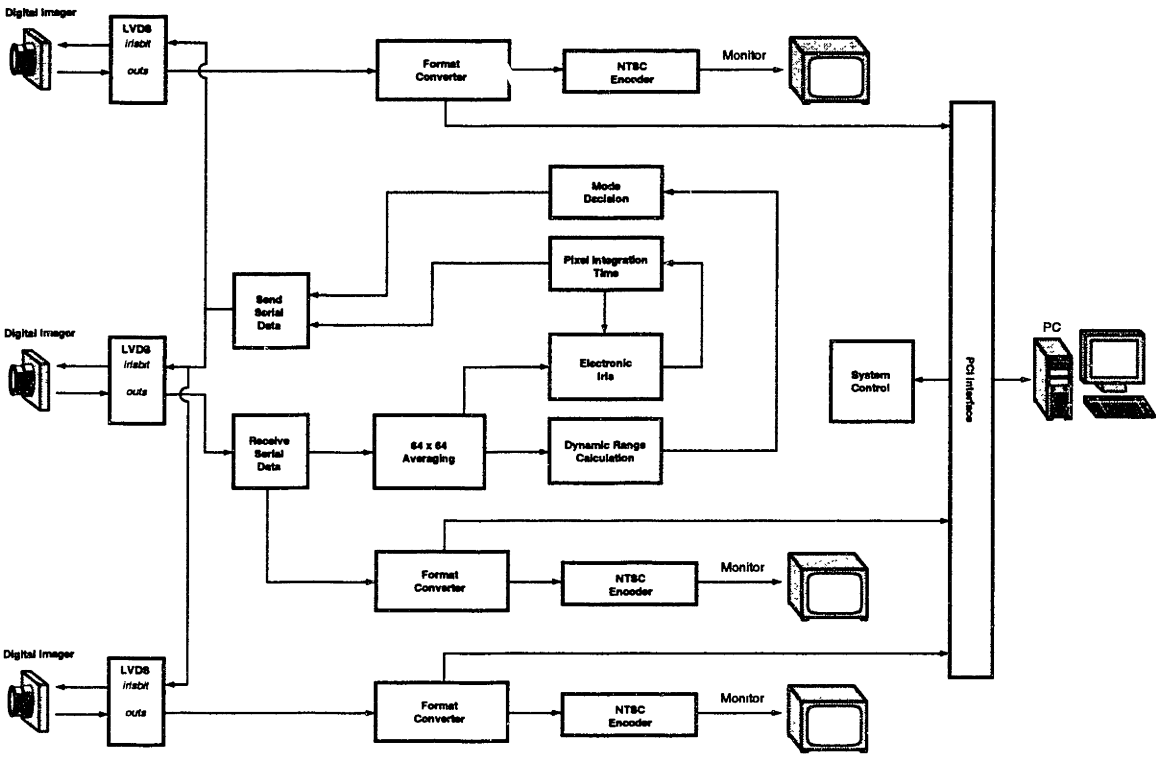

Figure 4.21: Block diagram of the 3-camera stereo vision system.

and darkest spots of the image in order to ensure that the algorithm is working properly.

4.5 3-Camera Board with Automatic Brightness Adaptation

The final 3-camera stereo vision system is a combination of the other boards described in this chapter. A block diagram of the system is given in Figure 4.21. The brightness adaptive control parameters of the imagers are all based on the center imager's data which ensures that there is good correlation between image data. The same output from the feedback loop is sent to all the imagers but the data from each imager is formatted separately. The control signals for the format converters and the interface to the GuPPI card are all shared between the blocks. Each imager's datapath contains its own memory buffer and SRAM controller.

Chapter 5

Results

The results from each stage of the project contributed to the final 3-camera stereo vision system. A discussion of the problems and the results of each stage of the project are included here. Section 5.1 addresses the performance achieved by the real-time imager demonstration. Section 5.2 discusses the results of the automatic brightness adaptive algorithm. Section 5.3 addresses the PCI interface. Finally, Section 5.4 discusses the completion of the 3-camera stereo vision project. All figures in this chapter are digital images generated by the imager itself.

5.1 Real-time Imager Demonstration

The two board design for the real-time imager demonstration was completed without any major problems. Figure 5.1 shows the fronts and backs of the circuit boards in the first two pictures and the assembled camera in the last picture. The major contributing factor to completing the project was the design and testing of the imager in stages. Many lines of VHDL code were written to program the imager and it would have been impossible to get everything to work properly without a way of testing each stage. The NTSC encoder was programmed and tested in a CPLD before the circuit board was made. The format converter and imager timing signals were thoroughly simulated. One of the more creative ways of initially testing the imager board was the implementation of a battery tester. The imager board must receive 5 volts in order to operate. Since a dual package opamp was used for the NTSC encoder there was one opamp left unused. By configuring the second opamp as a comparator, the scaled battery voltage could be compared against a zener diode. When the supply voltage fell below a desired level, the power LED indicator would start to blink. The signals used to cause the LED to blink were important timing signals for the pixel array. Correct operation of the CPLD and timing signals could easily be confirmed by simply turning down the supply voltage. After initial problems programming the CPLDs, and after correcting a data shuffling error, a real-time image could be displayed on the screen. A long period of fine tuning was necessary in order to get a very good quality image. Most of the problems were due to the offset in the CDS circuitry and the bias currents for the on-chip opamps. Also, the voltage level for the barrier function was finally set to the appropriate level. Some small white dots appeared in the image that were due

to digital noise near the opamp. To correct this, the traces on the printed circuit board were cut and rewired by hand. The last issue with the real-time imager demonstration was pixel lag. During the pixel reset period, a small proportion of charge from the previous integration period is left in the pixel. The imager produced slight image lag that was reduced by turning off the effect of the common gate amplifier. The voltage *cgbias* (see Chapter 2) was therefore set to the supply voltage.

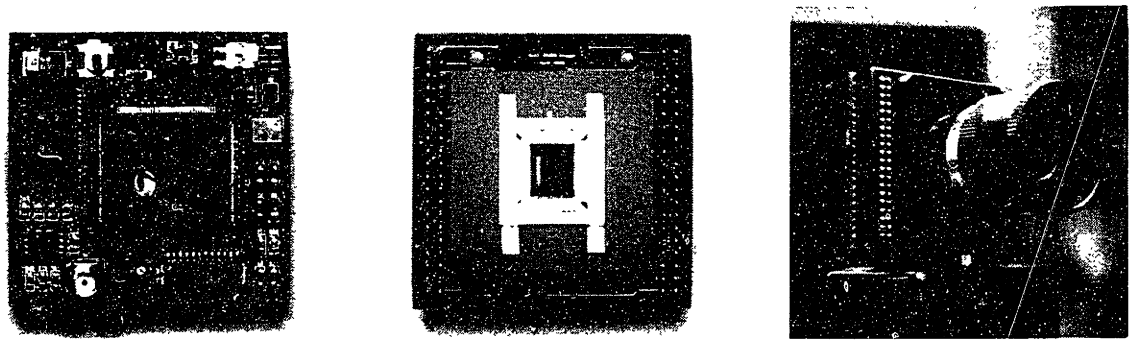

Figure 5.1: The real-time imager circuit boards and assembled camera.

A standard demonstration of the real-time NTSC camera consists of showing an image that requires a wide dynamic range and then switching between linear and logarithmic modes. The first image in Figure 5.2 shows the scene with the iris set so that the filament in the light bulb can be seen. Then the iris is set so that the sign can be read. Finally, the imager is switched to logarithmic compression and both the filament in the light bulb and the details in the sign are captured. Since the first real-time demonstration of the imager did not have a data interface for a PC, the later digital version of the imager provided the images for this figure.

Figure 5.2: Images captured with linear compression at two different iris settings.

5.2 Automatic Brightness Adaptation

The automatic brightness adaptive imaging test board, called *Autobright*, was the major testing platform for the project. Figure 5.3 shows the circuit board with all of the components in place. Figure 5.4 shows the components for the digital imager. The lens cover,

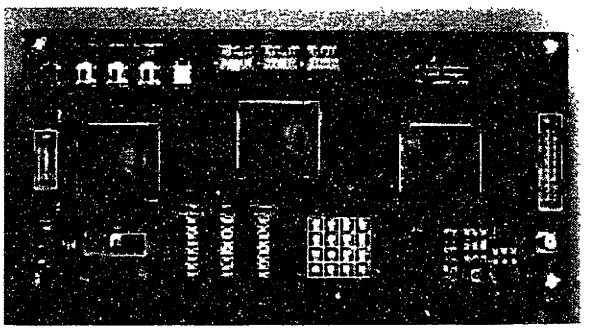

Figure 5.3: *Autobright* test board for auto-irising and compressing.

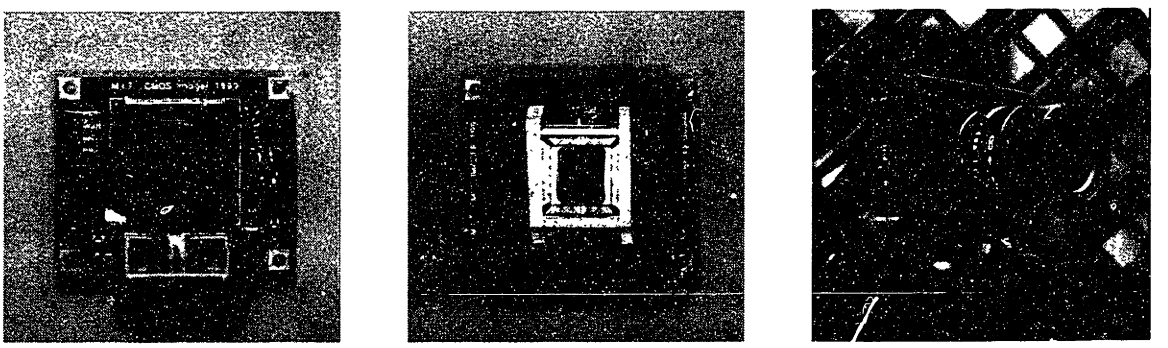

Figure 5.4: Components for the digital camera.

shown in Figure 5.5, is an improvement over the previous design because there are less internal reflections. The two board design was made to keep the overall size to a minimum. The debugging procedure for the circuit board was a long process. Most of the problems were found systematically by reprogramming the CPLDs with test code. Solder problems were a big source of error in the beginning. One of the circuit boards was fabricated with shorted traces.

The *Autobright* test board contained a digital output to a daughter board called *Tadpole*

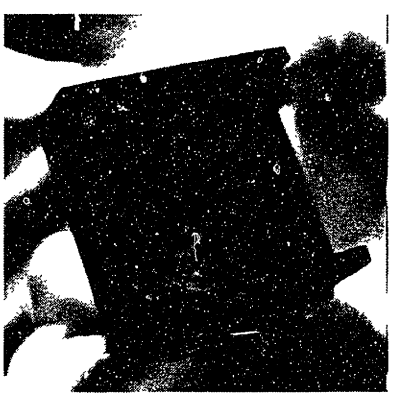

Figure 5.5: Lens cover for the digital imager that minimizes internal reflections.

which provided a path into the PC through the PCI bus. This enabled quick analysis and viewing of the captured images. The raw data was important for debugging purposes because absolute intensities were written to files and displayed on a computer monitor. The standard video monitor has automatic gain control and contrasting that make it difficult to understand what data the imager is producing.

The auto-irising feature worked as expected. The settling time for the frame average after a change in lighting conditions is much faster than the eye can perceive. Figure 5.6 shows three sample images that demonstrate the auto-iris. The first picture is an image taken after the iris has settled to steady state. The auto iris was then disabled for a small period while the light is turned on. This causes the average image intensity to go beyond half-scale. The auto-iris is then enabled which causes the frame average to move to exactly half-scale in the next frame.

The iris value which represents the pixel's charge integration time is updated every other frame. The reason for this is that the controller must wait an entire frame period after an iris value is applied before the image data corresponds with that iris value. The integration time specified by the iris value is set in the pixel array at readout time. Therefore, at the next readout time the pixel will have integrated the charge over the time interval specified during the last readout sequence. The controller simply ignores the imager's frame data that is generated while a new iris value is applied. However, at the end of the next frame a new iris value is immediately calculated if the frame average is not at half-scale. This iris value is then applied during the next two frames. Although the controller ignores every other frame, all frames are displayed or transmitted to the computer.

Using the method for calculating the iris value described in Section 4.1.3, the response time for the auto-iris should be $1/(framerate/2)$. The only delay in the response comes from waiting for the end of the integration period and for the entire frame data to be read out of the imager. This delay is effectively two frame periods. Testing at a rate of 60fps has confirmed that the iris value settles to within $\pm 1LSB$ in $1/30Hz \approx 33ms$.

Figure 5.6: Sequence of frames using auto-irising.

The automatic compression function also worked as expected. Figure 5.7 shows the two images with and without automatic compression. The feature is a little unstable in border line cases when the hysteresis value is not appropriate for the lighting conditions. The threshold values for entering and exiting the logarithmic compression mode are set on an 8-bit scale. The threshold values are compared against the pseudo dynamic range value for the

frame as described in Section 4.1.3. The linear threshold was set to 64 and the logarithmic threshold was set to 128. This gave enough hysteresis to avoid oscillations in nearly all cases. Occasionally, between the borders of the individual regions that are illustrated in Figure 4.3 in Section 4.1.3 on page 37, the imager can still oscillate between linear and logarithmic compression. Some ways to improve the the decision rule are discussed in Chapter 6.

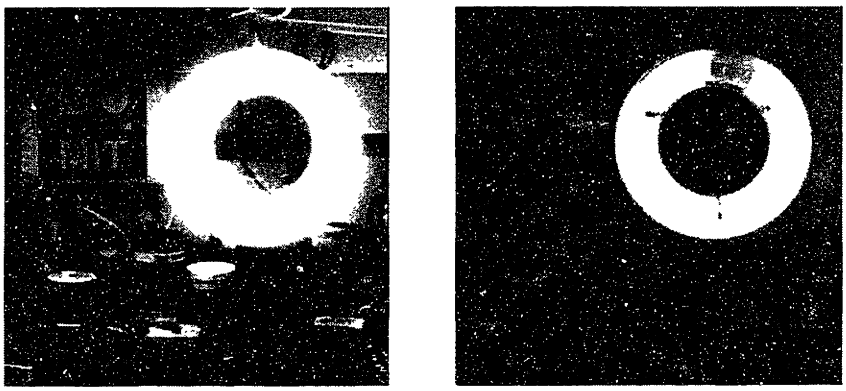

Figure 5.7: Sample images without and with auto compression.

In order to see some initial results of the imager in its intended environment, it was put to the test on a pair of headlights. Figure 5.8 shows the images captured with a small integration period, a long integration period, and finally with a logarithmic compression function.

5.3 PCI Interface

The original specification for the frame rate delivered to the PC was 60fps. The PCI interface can not yet handle this rate so only 30fps are currently delivered to the PC. This problem will be solved after more work has been done on the software system for communicating with the GuPPI card. Figure 5.9 shows the daughter card *Tadpole* that plugs into the GuPPI card.

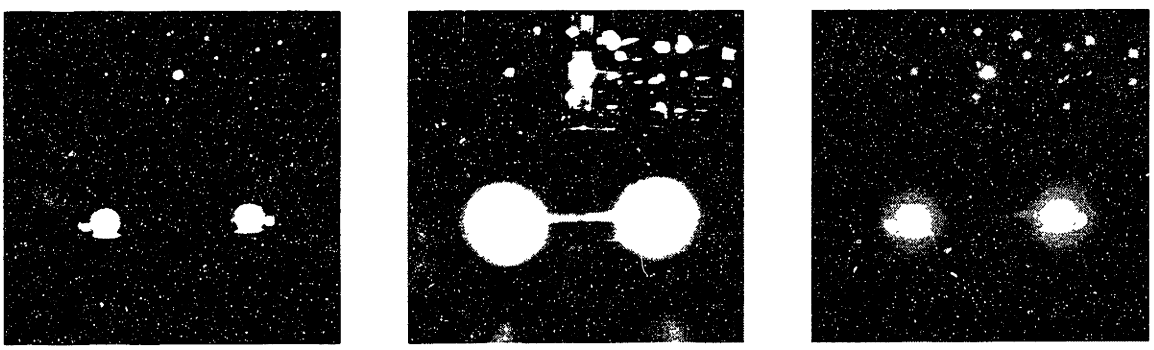

Figure 5.8: Headlight test.

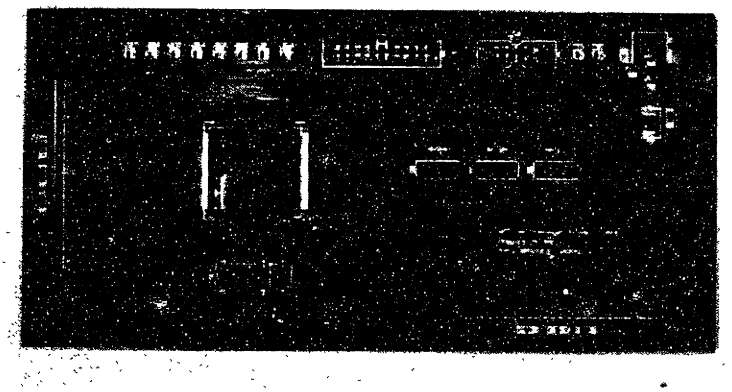

Figure 5.9: *Tadpole* test board for auto-irising and compressing.

5.4 3-Camera System

The final 3-camera printed circuit board which interfaces to the PC is still being fabricated. The board is essentially a combination of the *Autobright* and *Tadpole* circuit boards. Three imagers, plug into the back of the PC via twisted pair. The circuit board also has a standard NTSC output signal generated for each imager. Figure 5.10 shows how the three digital imagers are mounted to a vehicle for stereo processing. The image data is written into the RAM of the PC and displayed on the monitor via the interface shown in Figure 5.11. Although the automatic brightness adaptation algorithm is implemented in hardware, the user can still select between compression functions and choose to enable or disable auto-irising from the PC terminal. Various frame rate values can also be selected from the terminal.

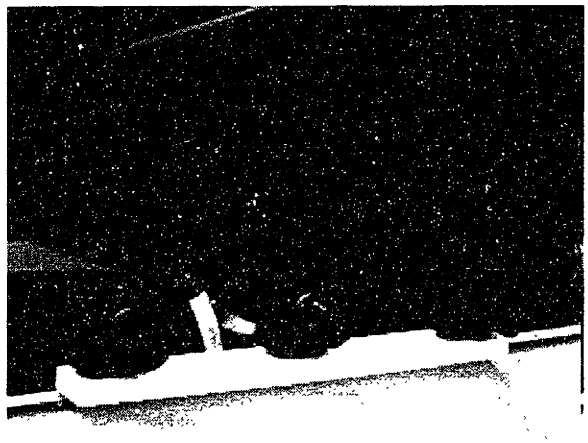

Figure 5.10: 3-camera stereo vision arrangement.

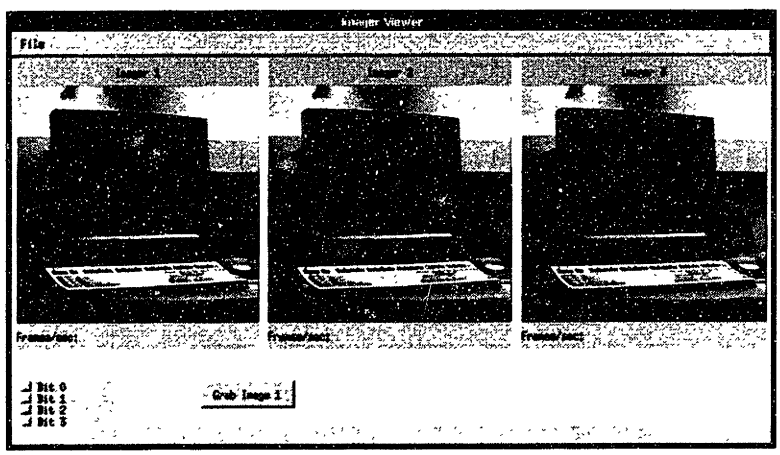

Figure 5.11: 3-camera viewer.

Chapter 6

Conclusion

This thesis has described the design and implementation of a 3-camera stereo vision system and the various stages of the project. This chapter summarizes the project and presents suggestions for future work.

6.1 Summary

The wide dynamic range image sensor with lateral overflow drain proves to be an effective architecture for machine vision applications where varying levels of illumination are expected. Many techniques have been shown to enhance the dynamic range of image sensors but few present system-level solutions. The objective of this project has been to create a system that adapts to the environment by providing high detail video images even in the most extreme conditions. The specific requirements for the project have been met through the design and implementation of both electronic iris control and automatic compression. A full stereo vision system has been designed and implemented from the front-end pixel array to the PCI interface for general purpose processing. In building the system from the sensor to the processor, many of the historic vision formatting limitations have been overcome. While the imager can be adjusted for any frame rate (and reconfigured on the fly), it can still provide data in NTSC format for a standard video monitor.

6.2 Future Work

The automatic dynamic range adjustment algorithm has been effective in demonstrating the potential for intelligent compression schemes. More work should be dedicated to the development of a robust solution. Both the targeting of the wide dynamic range image and the fitting of the compression scheme can be improved. One suggestion to improve the current algorithm is to run a sweeping box average or median across the image as shown in Figure 6.1. By comparing the brightest area to the darkest area, a wide dynamic range image can be targeted. This will eliminate the possibility of indecision at the boundary lines that may occur in the current algorithm. Future work may involve implementing the improved algorithm in a small pixel-parallel processing array like the one described by Gealow [9].

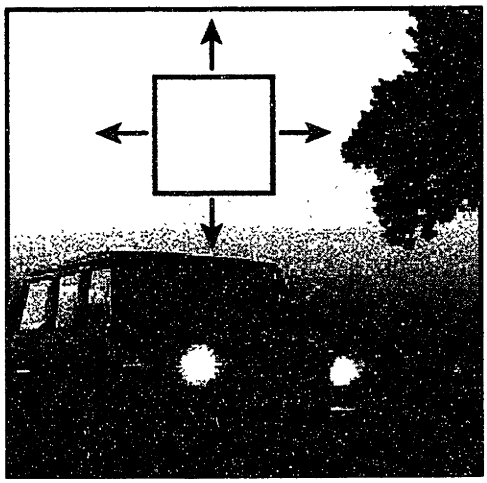

Figure 6.1: Sweeping Median Algorithm.

Another idea for improving the automatic compression scheme is to evaluate the frequency content of the image. The technique might be very similar to algorithms used for auto-focus lenses. There, the idea is to evaluate the luminance value of the video signal and adjust the lens until the highest frequency content is found. This ensures that the image is focused and sharp as opposed to soft and blurred. Rather than evaluating the pixel to pixel frequency content, the algorithm could evaluate the frequencies from region to region.

Whatever methods are used for determining the dynamic range of the image, it will probably be important to fit characteristic compression functions to different types of images. An image with a large dynamic range may have very few objects which fall in between brightest and darkest points in the image. Ideally, this image will be compressed differently than one with various light intensities spanning the entire dynamic range.

Finally, the 3-camera system should be put to use in an intelligent vehicle application. Figure 6.2 shows a block diagram of the conceptual setup for the demonstration of vehicle control based on stereo vision. The first stage of the project will involve creating and testing stereo algorithms. A powerful demonstration will be to show that the PC can make decisions based on how close objects come to the imagers. The PC could signal to the user that he has come too close for comfort by sending out a loud ear-piercing noise. A colorful image depth-map could also be designed and demonstrated in real-time. Once a robust set of algorithms have been created for the system, an interface to a small electric vehicle such a golf cart could be created to show adaptive cruise control, obstacle avoidance, etc.

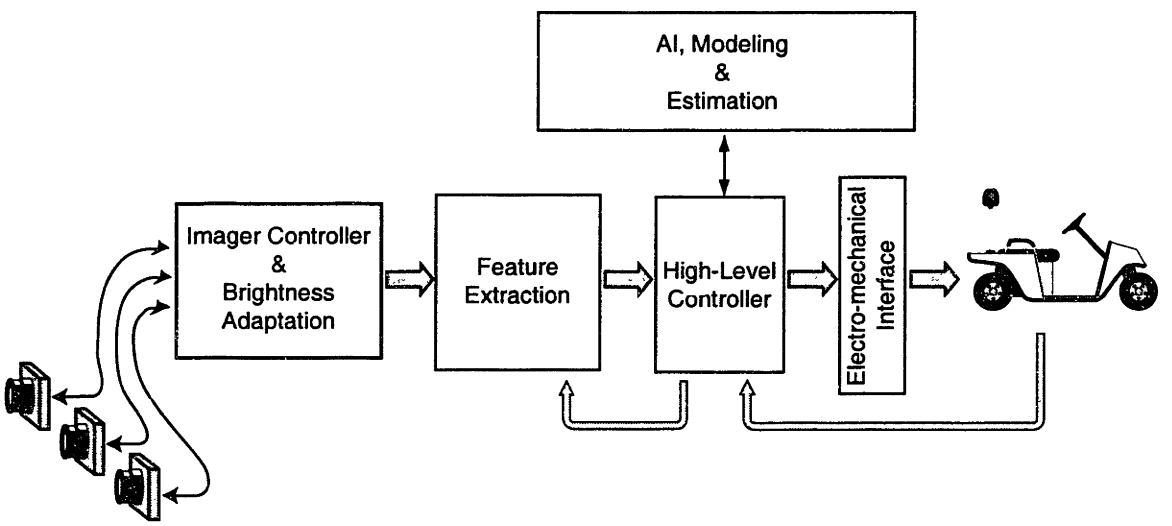

Figure 6.2: Intelligent vehicle demonstration system.

References

[1] M. Maurer, "A framework for flexible automation of semi-autonomous land vehicles," in *ISIC Intelligent Vehicle Control*, pp. 531–536, 1998.

[2] T. Kamada and K. Oikawa, "Amadeus: a mobile, autonoumous decentralised utility system for indoor transportation," in *IEEE Robotics and Automation*, pp. 2229–2236, May 1998.

[3] I. Masaki, R. Demir, and E. F. Crawley, "System design for intelligent transportation systems," in *Intelligent Vehicles Symposium*, pp. 323–326, 1996.

[4] H. Raza and P. Ioannou, "Vehicle following control design for automated highway," in *Vehicular Technology Conference*, pp. 904–908, May 1997.

[5] V. Gazi, M. Moore, and K. M. Passino, "Real-time control system software for intelligent system," in *ISIC Intelligent Vehicle Control*, pp. 102–107, 1998.

[6] Tao Xiping, Guo Muhe, and Ahang Bo, "A neural network approach to the elimination of road shadow for outdoor mobile robot," in *Intelligent Processing Systems*, pp. 1302–1306, October 1997.

[7] Ernst Dieter Dickmanns, "Road vehicle eyes for high precision navigation," in *High Precison Navigation*, pp. 329–336, 1995.

[8] Soo Kweon, Yue Bao, and N. Fujiwara, "Motion estimation from sequential image using correlation," in *ITSC Intelligent Transportation System*, pp. 775–780, November 1997.

[9] Jeffrey Carl Gealow, *An Integrated Computing Structure for Pixel-Parallel Image Processing.* PhD thesis, Massachusetts Institute of Technology, June 1997.

[10] Zubair Talib, "A real-time 256×256 pixel-parallel image processing system," Master's thesis, Massachusetts Institute of Technology, October 1998.

[11] Steven John Decker, *A Wide Dynamic Range CMOS Imager With Parallel On-Chip Analog-to-Digital Conversion.* PhD thesis, Massachusetts Institute of Technology, September 1997.

[12] M. Okutomi and S. Noguchi, "Extraction of road region using stereo images," in *Pattern Recognition*, pp. 853–856, August 1998.

[13] J. Kosecka, R. Blasi, C. J. Taylor, and J. Malik, "A comparative study of vision-based lateral control," in *Robotics and Automation*, pp. 1903–1908, May 1998.

[14] A. Kelly and A. Stentz, "Analysis of requirements for high speed rough terrain autonomous mobility," in *Robotics and Automation*, pp. 3326–3333, 1997.

[15] i-Sight, *Adaptive Sensitivity*, May 1999.
`http://www.i-sight.com/as1.htm`.

[16] Ginosar, Ran, Hilsenrath, Oliver, Zeevi, and Yehoshua, "Wide dynamic range camera," in *U.S Patent No. 5,144,442*, 1992.

[17] D. Yang, A. El Gamal, B. Fowler, and H. Tian, "A 640 x 512 cmos image sensor with ultra wide dynamic range floating point pixel level adc," in *ISSCC Digest of Technical Papers*, February 1999.

[18] K. Yamada, T. Nakano, and S. Yamamoto, "A vision sensor having an expanded dynamic range for autonomous vehicles," in *IEEE Transactions on Vehicular Technology*, pp. 47(1):332–341, February 1998.

[19] K. Yamada, T. Nakano, and S. Yamamoto, "Effectiveness of video camera dynamic range expansion for lane mark detection," in *IEEE Conference on Intelligent Transportation Systems*, pp. 584–588, November 1997.

[20] K. Yamada, T. Nakano, and S. Yamamoto, "Wide dynamic range vision sensor for autonomous vehicles," in *IEEE International Conference on Robotics and Automation*, pp. 1:770–775, May 1995.

[21] T. Delbruck and C.A. Mead, "Adaptive photoreceptor with wide dynamic range," in *IEEE International Symposium on Circuits and Systems*, pp. 4:339–342, 1994.

[22] M.D. Rowley, J.G. Harris, and Shao-Jen Lim, "A logarithmic photoreceptor incorporating lateral bipolar," in *IEEE International Symposium on Circuits and Systems*, pp. 3:1852–1855, 1997.

[23] F.J. Kub and H.C. Lin, "High dynamic range, low-noise floating-gate photosensor," in *International Electronic Devices Meeting*, pp. 919–922, December 1996.

[24] O. Vietze and P. Seitz, "Active pixels for image sensing with programmable, high dynamic range," in *Advanced technologies, Intelligent Vision*, pp. 15–18, October 1995.

[25] A. Tanabe, Y. Kudoh, Y. Kawakami, K. Masabuchi, S. Kawai, T. Yamada, M. Morimoto, K. Arai, K. Hatano, M. Furumiya, Y. Nakashiba, n. Mutoh, K. Orihara, and H. Teranishi, "Dynamic range improvement by narrow-channel effect,"

[26] T. Hamamoto, K. Aizawa, and M. Hatori, "Motion adaptive image sensor," in *Asia and South Pacific Design Automation Conference*, pp. 343–344, February 1998.

[27] M. Bohm, T. Lule, H. Fischer, J. Schulte, B. Schneider, S. Benthien, F. Blecher, S. Coors, A. Eckhardt, H. Keller, P. Rieve, K. Seibel, M. Sommer, and J. Sterzel, "Design and fabrication of a high dynamic range image sensor," in *Symposium on VLSI Circuits*, pp. 202–203, 1998.

[28] Ray Furlong, "Troubleshooting camcoder zoom lenses," in *Electronics Now*, pp. 76–80, March 1997.

[29] S. Sakaue, M. Nakayama, A. Tamura, and S. Maruno, "Adaptive gama processing of the video cameras for the expansion of the dynamic range," in *IEEE Transactions on Consumer Electronics*, pp. 41(3):555–562, August 1995.

[30] Michael Ismert, "Making commodity pcs fit for signal processing," *Software Devices and Systems Group*, 1998.
`http://www.sds.lcs.mit.edu`.

[31] Michael Ismert, "Guppi board description," *Software Devices and Systems Group*, 1998.
`http://www.sds.lcs.mit.edu/SpectrumWare/guppi.html`.

Appendix A

Real-Time Imager Demonstration

A.1 VHDL

The VHDL files used for this board are located in:

```
/homes/kfife/Imager1/24.5MHz/
    imsync.vhd
    format.vhd
    image.vhd
    inout.vhd
    ntsc.vhd

    Project name: warp.pfg
```

These files are used to generate jedec files for two devices. The two top level designs are held in imsync.vhd and format.vhd. The lower level files are packages that are included in the top level designs.

A.1.1 Top Level VHDL - imsync.vhd

Location:

`/homes/kfife/Imager1/24.5MHz/imsync.vhd`

```
Library ieee;   --this file uses a 49MHz clock and divides
                --it by 2 to get the right NTSC freq
Use ieee.std_logic_1164.all;

Entity imsync IS PORT (
        clock,reset: IN std_logic;

        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;
        Aselect, Bselect: BUFFER std_logic;
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
        mode: IN std_logic;              --integration or linear mode
        colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
```

```
        phi5,phi6: OUT std_logic;
        flip: BUFFER std_logic;
        dataready,startframe: OUT std_logic;

---LOW BATTERY LOGIC

        vlevel: IN std_logic;
        led: OUT std_logic;
---CLOCKS
        cin: IN std_logic;
        clk: BUFFER std_logic;
        c: OUT std_logic; --goes to pin 98
        clkin: IN std_logic; --comes in on pin 99 dedicated to clocks
---MODE LED

        modeled: OUT std_logic;
---
        hsync,hblank: BUFFER std_logic;
        blnk: OUT std_logic;
        startr,startf: OUT std_logic);
---
attribute pin_avoid of imsync:entity is "6 46 76 116";
---------------------------------sclk,smode,SDO,SDI

attribute pin_numbers of imsync:entity is "clock:19 " &
"reset:59 " &
"rsin:143 " &

"swsel3:144 " &
"swsel2:145 " &
"swsel1:146 " &

"colsel(2):70 " &
"colsel(1):67 " &
"colsel(0):65 " &

"clkBB:23 " &
"clkA:24 " &
"phi6:25 " &
"phi5:26 " &
"phi4:27 " &
"phi3:28 " &
"phi2:29 " &
"phi1:30 " &

"phi0bar:148 " &

"phi0:147 " &

"Aselect:32 " &
"Bselect:33 " &
"isolate:34 " &
"sample1:35 " &
"sample2:36 " &
"clkr:37 " &
"clkb:38 " &

"flip:149 " &

"hsync:2 " &
"hblank:3 " &
"blnk:4 " &

"dataready:89 " &
"startframe:63 " &

"startr:18 " &
"startf:13 " &
"mode:91 " &

"vlevel:92 " &
"led:93 " &

"modeled:94 " &

"cin:77 " &
"clk:79 " &
"c:98 " &
"clkin:99 ";

End imsync;

USE WORK.std_arith.ALL;

Use work.ntscpkg.all;
Use work.imagepkg.all;
```

```
ARCHITECTURE behavior OF imsync IS

        SIGNAL flash: std_logic;

BEGIN

image0: im PORT MAP(
        clock,reset,
        clkBB,clkA,swsel1,swsel2,swsel3,
        phi1,phi2,phi3,phi4,
        Aselect, Bselect,
        phi0,phi0bar,
        isolate,
        clkb,clkr,
        rsin,
        mode,               --integration or linear mode
        colsel,
        sample1, sample2,
        phi5,phi6,
        flip,
        flash,
        dataready,startframe);

ntsc0: ntsc PORT MAP(
        clock,reset,
        hsync,hblank,
        startr,startf);

led <= flash when vlevel = '0' else '1';

modeled <= '1' when mode = '1' else '0';

--blanking for DAC--
blnk <= '1' when hblank = '0' else '0';
--------------------
c <= cin;

process begin
Wait Until clkin = '1';
clk <= not clk;
end process;

END behavior;
```

A.1.2 Imager Timing and Barrier Function - image.vhd

Location:

/homes/kfife/Imager1/24.5MHz/image.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity im IS PORT (
        clock,reseti: IN std_logic;                     --reset active low
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;          --ADC timing
        Aselect, Bselect: BUFFER std_logic;             --ADC selector for pad driver
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
        mode: IN std_logic;               --integration or linear mode
        colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
        phi5,phi6:OUT std_logic;
        flip: BUFFER std_logic;
        flash: OUT std_logic;
        dataready,startframe: OUT std_logic);   --used to say when to start a frame and when data is ready
End im;

USE WORK.std_arith.ALL;

ARCHITECTURE fsmim OF im IS

        SIGNAL i: integer range 0 to 49;             --i is used to divide the clock by 50.
        SIGNAL r: integer range 0 to 255;            --r is the number of rows

        SIGNAL count: std_logic_vector(5 downto 0);
        SIGNAL slow: std_logic_vector(2 downto 0);
```

```
        SUBTYPE v3 is std_logic_vector(2 downto 0);
        alias switch: v3 is count(2 downto 0);

        SIGNAL phih1,phih2,phih3,phih4: boolean;

BEGIN

PROCESS (clock,i,switch,reseti)

BEGIN
        IF      reseti = '0' THEN
                i <= 0;
                r <= 0;
                count <= "000000";
                flip <= '0';

        ELSIF clock'EVENT AND clock = '1' THEN
                if i = 49 then              --divide down the system clock
                        i <= 0;

                        --phi1-4
                        if (switch = "111") or (switch = "000")then
                        phih3 <= TRUE; else phih3 <= FALSE; end if;

                        if (switch = "001") or (switch = "010")then
                        phih4 <= TRUE; else phih4 <= FALSE; end if;

                        if (switch = "011") or (switch = "100")then
                        phih1 <= TRUE; else phih1 <= FALSE; end if;

                        if (switch = "101") or (switch = "110")then
                        phih2 <= TRUE; else phih2 <= FALSE; end if;

                        --Aselect and Bselect
                        if ( count = "000000") then
                        Bselect <= '1'; end if;
                        if ( count = "011111") then
                        Bselect <= '0'; end if;
                        if ( count = "100000") then
                        Aselect <= '1';  end if;
                        if ( count = "111111") then
                        Aselect <= '0'; end if;

                        --sample1 and sample2 and isolate
                        if ( count = "100000") then
                        sample2 <= '0'; end if;
                        if ( count = "100001") then
                        sample1 <= '1'; end if;
                        if ( count = "101101") then
                        sample1 <= '0';  end if;
                        if ( count = "101110") then
                        sample2 <= '1'; end if;

                        if ( count = "100001") then
                        isolate <= '1'; end if;
                        if ( count = "111011") then
                        isolate <= '0'; end if;

                        --clkb
                        if ( count = "101101") then
                        clkb <= '1';
                        else clkb <= '0';
                        end if;

                        --clkr
                        if ( count = "100000") then
                        clkr <= '1';
                        else clkr <= '0';
                        end if;

                        if count = "111111" then
                        count <= "000000";
                        if r = 255 then          --counting rows
                                r <= 0;
                                flip <= NOT flip;  --flip SRAMS
                                slow <= slow + 1;
                        else r <= r + 1;
                        end if;
                        else count <= count + 1;
                        end if;

                else i <= i + 1;
                end if;

        END IF;
```

```
END PROCESS;

flash <= slow(2);

------------non-overlapping phis----------
phi1 <= '1' when phih1 and not phih4 else '0';
phi2 <= '1' when phih2 and not phih1 else '0';
phi3 <= '1' when phih3 and not phih2 else '0';
phi4 <= '1' when phih4 and not phih3 else '0';
------------------------------------------

clkA <= '0' when switch = "000" else '1';
clkBB <= '0' when switch = "100" else '1';

phi0 <= '1' when (count(4 downto 0) > "11011") else '0';
phi0bar <= not phi0;

--rsin
rsin <= '1' when r = 255 else '0';

--phi5, phi6
phi5 <= Aselect;
phi6 <= Bselect;

swsel2 <= switch(0);
swsel1 <= switch(1);
swsel3 <= switch(2);

----------------------------------barrier function
process(r,mode,clock,colsel)
begin

IF clock'EVENT AND clock = '1' THEN

if mode = '1' then     --integration mode; approximating log function
        case r is
                when 255 => colsel <= "000"; --highest potential(reset = 5v)
                when 0   => colsel <= "001";
                when 127 => colsel <= "010";
                when 191 => colsel <= "011";
                when 223 => colsel <= "100";
                when 239 => colsel <= "101";
                when 247 => colsel <= "110";
                when 251 => colsel <= "111"; --lowest potential( = 2.5v)
                when others => colsel <= colsel;
        end case;
else
        case r is      --linear mode
                when 255 => colsel <= "000";
                when 0   => colsel <= "111";
                when others => colsel <= colsel;
        end case;

end if;

END IF;

end process;
-----------------------------------------------------------

-----------------------------------interface-----------------------------------

dataready <= '1' when i = 9 else '0';
startframe <= '1' when r = 0 else '0';

-------------------------------------------------------------------------------

END fsmim;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE imagepkg IS
COMPONENT im PORT (
        clock,reseti: IN std_logic;
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;
        Aselect, Bselect: BUFFER std_logic;
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
        mode: IN std_logic;               --integration or linear mode
        colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
        phi5,phi6: OUT std_logic;
        flip: BUFFER std_logic;
```

```
        flash: OUT std_logic;
        dataready,startframe: OUT std_logic);
END COMPONENT;
END imagepkg;
```

A.1.3 Top Level VHDL - format.vhd

Location:

/homes/kfife/Imager1/24.5MHz/format.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity systop IS PORT (
        clock,reset: IN std_logic;
        WE,OE: OUT std_logic;
        addr: BUFFER std_logic_vector(15 downto 0);
        chip: IN std_logic_vector(0 to 31);
        IO: INOUT std_logic_vector(7 downto 0);
        dataready,startframe: IN std_logic;
        ----------------------------
        startf,startr: IN std_logic;
        WE1,OE1: OUT std_logic;
        addr1: BUFFER std_logic_vector(15 downto 0);
        IO1: INOUT std_logic_vector(7 downto 0);
        pixout: BUFFER std_logic_vector(7 downto 0);

        flip: IN std_logic);

attribute pin_avoid of systop:entity is "6 46 76 116";
-----------------------------------sclk,smode,SDO,SDI

attribute pin_numbers of systop:entity is "clock:19 " &
"reset:59 " &
"WE:11 " &
"OE:12 " &

"addr(0):143 " &
"addr(1):144 " &
"addr(2):145 " &
"addr(3):146 " &
"addr(4):147 " &
"addr(5):148 " &
"addr(6):149 " &
"addr(7):150 " &
"addr(8):78 " &
"addr(9):48 " &
"addr(10):36 " &
"addr(11):16 " &
"addr(12):85 " &
"addr(13):87 " &
"addr(14):158 " &
"addr(15):137 " &

"chip(0):63 " &
"chip(1):58 " &
"chip(2):64 " &
"chip(3):57 " &
"chip(4):65 " &
"chip(5):56 " &
"chip(6):66 " &
"chip(7):55 " &

"chip(8):67 " &
"chip(9):54 " &
"chip(10):68 " &
"chip(11):53 " &
"chip(12):69 " &
"chip(13):52 " &
"chip(14):70 " &
"chip(15):51 " &

"chip(16):91 " &
"chip(17):30 " &
"chip(18):92 " &
"chip(19):29 " &
"chip(20):93 " &
"chip(21):28 " &
"chip(22):94 " &
```

```
"chip(23):27 " &

"chip(24):95 " &
"chip(25):26 " &
"chip(26):96 " &
"chip(27):25 " &
"chip(28):97 " &
"chip(29):24 " &
"chip(30):98 " &
"chip(31):23 " &

--original pins were changed for the correct format--
--"chip(0):23 " &
--"chip(1):24 " &
--"chip(2):25 " &
--"chip(3):26 " &
--"chip(4):27 " &
--"chip(5):28 " &
--"chip(6):29 " &
--"chip(7):30 " &
--
--"chip(8):51 " &
--"chip(9):52 " &
--"chip(10):53 " &
--"chip(11):54 " &
--"chip(12):55 " &
--"chip(13):56 " &
--"chip(14):57 " &
--"chip(15):58 " &
--
--"chip(16):63 " &
--"chip(17):64 " &
--"chip(18):65 " &
--"chip(19):66 " &
--"chip(20):67 " &
--"chip(21):68 " &
--"chip(22):69 " &
--"chip(23):70 " &
--
--"chip(24):91 " &
--"chip(25):92 " &
--"chip(26):93 " &
--"chip(27):94 " &
--"chip(28):95 " &
--"chip(29):96 " &
--"chip(30):97 " &
--"chip(31):98 " &

"IO(0):32 " &
"IO(1):42 " &
"IO(2):72 " &
"IO(3):82 " &
"IO(4):112 " &
"IO(5):7 " &
"IO(6):114 " &
"IO(7):118 " &

"dataready:2 " &
"startframe:3 " &

"startf:4 " &
"startr:5 " &

"WE1:13 " &
"OE1:14 " &

"addr1(0):122 " &
"addr1(1):123 " &
"addr1(2):124 " &
"addr1(3):125 " &
"addr1(4):126 " &
"addr1(5):127 " &
"addr1(6):128 " &
"addr1(7):129 " &
"addr1(8):79 " &
"addr1(9):49 " &
"addr1(10):37 " &
"addr1(11):17 " &
"addr1(12):86 " &
"addr1(13):88 " &
"addr1(14):159 " &
"addr1(15):138 " &

"IO1(0):33 " &
"IO1(1):43 " &
"IO1(2):73 " &
```

```
"IO1(3):83 " &
"IO1(4):113 " &
"IO1(5):8 " &
"IO1(6):115 " &
"IO1(7):119 " &


"pixout(0):152 " &
"pixout(1):153 " &
"pixout(2):131 " &
"pixout(3):132 " &
"pixout(4):103 " &
"pixout(5):104 " &
"pixout(6):38 " &
"pixout(7):39 " &

"flip:15 ";

End systop;

Use work.inopkg.all;

ARCHITECTURE behavioral OF systop IS

BEGIN
ino0: ino PORT MAP(
                clock,reset,
                WE,OE,
                addr,
                chip,
                IO,
                dataready,startframe,

                startf,startr,
                WE1,OE1,
                addr1,
                IO1,
                pixout,

                flip);
END behavioral;
```

A.1.4 Format Conversion - inout.vhd

Location:

/homes/kfife/Imager1/24.5MHz/inout.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity ino IS PORT (
        clock: IN std_logic;
        reset: IN std_logic;  --active low
        WE,OE: OUT std_logic;
        addr: BUFFER std_logic_vector(15 downto 0);
        chip: IN std_logic_vector(0 to 31);
        IO: INOUT std_logic_vector(7 downto 0);
        pix,start: IN std_logic;            --start indicates begining of frame
                                            --pix indicates when a set of 32 bits from the imager are ready
        startf,startr: IN std_logic;   --startf causes pixels to be output at begining of frame
        WE1,OE1: OUT std_logic;        --startr cause pixels to be output at begining of row
        addr1: BUFFER std_logic_vector(15 downto 0);
        IO1: INOUT std_logic_vector(7 downto 0);
        pixout: BUFFER std_logic_vector(7 downto 0);

        flip: IN std_logic);   --flip switches SRAM parameters. ie- addr,WE,OE,IO
End ino;

USE WORK.std_arith.ALL;

ARCHITECTURE fsmino OF ino IS
        TYPE states IS (idle,doread,dowrite,begins);
        SIGNAL state : states;

        SUBTYPE v17 is std_logic_vector(16 downto 0);
        SUBTYPE v8 is std_logic_vector(7 downto 0);
        SUBTYPE v2 is std_logic_vector(1 downto 0);
        SUBTYPE v3 is std_logic_vector(2 downto 0);
```

```
        SIGNAL row_prime_part_n_inc_pac: v17;

        alias pac: v3 is                row_prime_part_n_inc_pac(2 downto 0);
        alias inc: v2 is                row_prime_part_n_inc_pac(4 downto 3);
        alias n: std_logic is           row_prime_part_n_inc_pac(5);
        alias part: v2 is               row_prime_part_n_inc_pac(7 downto 6);
        alias prime: std_logic is       row_prime_part_n_inc_pac(8);
        alias row: v8 is                row_prime_part_n_inc_pac(16 downto 9);

        SIGNAL buff: std_logic_vector(3 downto 0);
---------------------------------
        TYPE states1 IS (rowwait, dorow);
        SIGNAL state1 : states1;

        SIGNAL count18: std_logic_vector(17 downto 0);

        alias row1: std_logic_vector(7 downto 0) is count18(17 downto 10);
        alias pac1: std_logic_vector(2 downto 0) is count18(9 downto 7);
        alias inc1: std_logic_vector(1 downto 0) is count18(5 downto 4);
        alias prime1: std_logic is count18(3);
        alias part1: std_logic_vector(1 downto 0)is count18(1 downto 0);

        SIGNAL startflag: std_logic;
        SIGNAL top: integer range 4 to 7;

        SIGNAL pixel: std_logic_vector(7 downto 0);

-------statemachine for input formatting-----------------
BEGIN
PROCESS (clock,pix,inc,n,part,prime,row,start,reset)
BEGIN

IF (reset = '0') THEN
        state <= begins;

        ELSIF clock'EVENT AND clock = '1' THEN
                CASE state IS
                        WHEN idle =>    IF (pix = '1') THEN
                                        state <= doread;
                                        END IF;
                        WHEN doread => state <= dowrite;
                        WHEN dowrite => IF (row_prime_part_n_inc_pac = "1111111111111111") THEN
                                        state <= begins;
                                        ELSIF (pac = "111") THEN
                                        state <= idle;
                                        ELSE
                                        state <= doread;
                                        END IF;
                                        row_prime_part_n_inc_pac <= row_prime_part_n_inc_pac + 1;
                        WHEN begins =>  IF (start = '1') THEN
                                        state <= idle;
                                        END IF;
                        WHEN OTHERS => state <= idle;
                END CASE;
        END IF;
END PROCESS;
-----------------------------------------------------------------

-------controlling WE and OE--------------------
PROCESS (state,flip,state1,count18)
BEGIN

if flip = '0' then
        if (state = dowrite) then
                WE <= '0';
        else WE <= '1';
        end if;

        if (state = doread) then
                OE <= '0';
        else OE <= '1';
        end if;

        if (state1 = dorow) then        ---from read process
        OE1 <= '0';
        else
        OE1 <= '1';
        end if;
        WE1 <= '1';
else
        if (state = dowrite) then
                WE1 <= '0';
        else WE1 <= '1';
        end if;

        if (state = doread) then
```

```
                OE1 <= '0';
        else OE1 <= '1';
        end if;

        if (state1 = dorow) then          ---from read process
        OE <= '0';
        else
        OE <= '1';
        end if;
        WE <= '1';
end if;
END PROCESS;
-----------------------------------------------

------------controlling IO ports----------------
PROCESS (state,clock,buff,IO,IO1,flip)
BEGIN
IF clock'EVENT AND clock = '1' THEN
        if flip = '0' then
                IF (state = doread) THEN  -- read out of SRAM and put the value in buff
                buff <= IO(3 downto 0);
                END IF;
        else
                IF (state = doread) THEN  -- read out of SRAM1 and put the value in buff
                buff <= IO1(3 downto 0);
                END IF;
        end if;
END IF;
END PROCESS;

PROCESS (state,buff,chip,flip,pac,clock)
variable index0: integer range 0 to 31;
variable index1: integer range 0 to 31;
variable index2: integer range 0 to 31;
variable index3: integer range 0 to 31;
BEGIN
index0 := to_integer(pac) * 4;
index1 := to_integer(pac) * 4 + 1;
index2 := to_integer(pac) * 4 + 2;
index3 := to_integer(pac) * 4 + 3;

IF (clock = '0') THEN  --output gets ored with clock so that there is no contention.
        if flip = '0' then
        IO1 <= "ZZZZZZZZ";                      --allows IO1 to be used as an input during this time
                IF (state = dowrite) THEN       --read out of SRAM and put the value in buff
                IO <= buff & chip(index0) & chip(index1) & chip(index2) & chip(index3);
                        --combine outputs of chip with the other four bits
                ELSE            --in buff and write to SRAM
                IO <= "ZZZZZZZZ";
                END IF;
        else
        IO <= "ZZZZZZZZ";
                IF (state = dowrite) THEN
                IO1 <= buff & chip(index0) & chip(index1) & chip(index2) & chip(index3);
                ELSE
                IO1 <= "ZZZZZZZZ";
                END IF;
        end if;
ELSE IO <= (others => 'Z'); IO1 <= (others => 'Z');
END IF;

END PROCESS;
-----------------------------------------

--------------RASTER------OUTPUT----------------
PROCESS (clock,state1,startf,IO1,pixel,startr,flip)

        VARIABLE bot: integer range 0 to 3:= top - 4;

BEGIN
        IF      (startf = '1') THEN
                state1 <= rowwait;
                count18 <= (others => '0');

        ELSIF clock'EVENT AND clock = '1' THEN

                CASE state1 IS

                        WHEN dorow =>   if flip = '0' then

                                        CASE part1 IS    -- read out of SRAM
                                                WHEN "00" => pixel(7 downto 6) <= IO1(top) & IO1(bot);
                                                WHEN "01" => pixel(5 downto 4) <= IO1(top) & IO1(bot);
                                                WHEN "10" => pixel(3 downto 2) <= IO1(top) & IO1(bot);
                                                WHEN "11" => pixel(1 downto 0) <= IO1(top) & IO1(bot);
                                                WHEN OTHERS => pixel <= pixel;
```

```
                                        END CASE;
                                        else
                                        CASE part1 IS    -- read out of SRAM
                                                WHEN "00" => pixel(7 downto 6) <= IO(top) & IO(bot);
                                                WHEN "01" => pixel(5 downto 4) <= IO(top) & IO(bot);
                                                WHEN "10" => pixel(3 downto 2) <= IO(top) & IO(bot);
                                                WHEN "11" => pixel(1 downto 0) <= IO(top) & IO(bot);
                                                WHEN OTHERS => pixel <= pixel;
                                        END CASE;

                                        end if;

                                        CASE part1 IS    -- read out of SRAM
                                                WHEN "00" => pixout <= pixel;
                                                WHEN OTHERS => pixout <= pixout;
                                        END CASE;

                                        IF (count18(9 downto 0) = "1111111111") THEN
                                        state1 <= rowwait;
                                        ELSE
                                        state1 <= dorow;
                                        END IF;

                                        count18 <= count18 + 1;

                        WHEN rowwait => count18 <= count18;
                                        pixout <= pixel;
                                        IF startr = '1' THEN
                                        state1 <= dorow;
                                        END IF;

                        WHEN OTHERS => state1 <= rowwait;
                END CASE;
        END IF;
END PROCESS;


PROCESS(count18)
BEGIN

IF count18(6) = '0' THEN
        IF count18(2) = '0' THEN
        top <= 7;
        ELSE top <= 6;
        END IF;
ELSE
        IF count18(2) = '0' THEN
        top <= 5;
        ELSE top <= 4;
        END IF;
END IF;
END PROCESS;

----------------output addressing----------------------
process(flip,row,prime,part,pac,inc,row1,prime1,part1,pac1,inc1)
begin
if flip = '0' then
addr <= row & prime & part & pac & inc;
addr1 <=  row1 & prime1 & part1 & pac1 & inc1;
else
addr1 <= row & prime & part & pac & inc;
addr <=  row1 & prime1 & part1 & pac1 & inc1;
end if;
end process;
-----------------------------------------------------

END fsmino;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE inopkg IS
COMPONENT ino PORT (
        clock: IN std_logic;
        reset: IN std_logic;
        WE,OE: OUT std_logic;
        addr: BUFFER std_logic_vector(15 downto 0);
        chip: IN std_logic_vector(0 to 31);
        IO: INOUT std_logic_vector(7 downto 0);
        pix,start: IN std_logic;            --start indicates begining of frame
                                            --pix indicates when a set of 32 bits from the imager are ready
        startf,startr: IN std_logic;    --startf causes pixels to be output at begining of frame
        WE1,OE1: OUT std_logic;         --starfr cause pixels to be output at begining of row
        addr1: BUFFER std_logic_vector(15 downto 0);
        IO1: INOUT std_logic_vector(7 downto 0);
        pixout: BUFFER std_logic_vector(7 downto 0);
```

```
      flip: IN std_logic);   --flip switches SRAM parameters. ie- addr,WE,OE,IO
END COMPONENT;
END inopkg;
```

A.1.5 NTSC encoder - ntsc.vhd

Location:

/homes/kfife/Imager1/24.5MHz/ntsc.vhd

```
--This is code to generate an interlaced ntsc encoder. The
--non-interlaced version version is better and more efficent
Library ieee;
Use ieee.std_logic_1164.all;
Entity ntsc IS PORT (
        clock,reset:    IN std_logic;   --reset active low
        hsync,hblank:   BUFFER std_logic;
        startr,startf:  OUT std_logic);
End ntsc;

USE WORK.std_arith.ALL;
USE WORK.int_math.ALL;

ARCHITECTURE fsmntsc OF ntsc IS
        SIGNAL count: integer range 0 to 1569;
        SIGNAL line: integer range 0 to 543;
BEGIN

PROCESS (clock,count,reset)

BEGIN

        IF      reset = '0' THEN
                count <= 0;
                line <= 0;
        ELSIF clock'EVENT AND clock = '1' THEN
        -- odd goes from 0 to 242, even goes from 271 to 513

                IF (NOT( (line > 259 AND line < 270)or(line > 531 AND line < 543) ) and count = 1559) THEN
                count <= 0;
                line <= line + 1;

                ELSIF ((line = 542) AND (count = 1559)) THEN  --reset on the end of line 542
                count <= 0;
                line <= 0;

                ELSIF (count = 1569) THEN --kills 10 clks for the above intervals. (200 total killed clks)
                count <= 0;
                line <= line + 1;

                --Eq and Ser pulses
                ELSIF ((line > 241 AND line < 260) OR (line > 513 AND line < 532)) AND count = 779 THEN
                count <= 0;
                line <= line + 1;

                ELSE count <= count + 1;
                END IF;

                CASE count IS

                WHEN 0 =>       hsync <= '0';   --start hsync pulse

                WHEN 123 =>     IF NOT ((line > 242 AND line < 261) OR (line > 513 AND line < 532))   THEN
                                        hsync <= '1';  --end hsync pulse
                                ELSE hsync <= hsync;
                                END IF;

                WHEN 57 =>      IF (line > 242 AND line < 249) OR  (line > 254 AND line < 261) OR (line > 513 AND line < 520) OR  (line > 525 AND line < 532) THE
                                hsync <= '1';   --eq pulse width
                                ELSE hsync <= hsync;
                                END IF;

                WHEN 659 =>     IF (line > 248 AND line < 255) OR (line > 519 AND line < 526) THEN  --ser width
                                        hsync <= '1';
                                ELSE hsync <= hsync;
                                END IF;

                WHEN OTHERS => hsync <= hsync;
                END CASE;
```

```
            CASE count IS

            WHEN 385 =>     IF NOT ((line > 242 AND line < 272) OR (line > 513 AND line < 543)) THEN
                            hblank <= '1';  --active video after back porch
                            ELSE hblank <= '0';
                            END IF;

            WHEN 779 =>     IF line = 271 THEN   --(1228/2 + 120 + 114 - 1 = 877) set to 779
                            hblank <= '1';  --active video again (top of screen)
                            ELSIF line = 242 THEN   --front porch going into ser
                            hblank <= '0';
                            ELSE hblank <= hblank;
                            END IF;

            WHEN 1409 =>    hblank <= '0';  --front porch start

            WHEN OTHERS => hblank <= hblank;
            END CASE;

            CASE count IS

            WHEN 379 =>     IF NOT ((line > 242 AND line < 272) OR
                            (line > 513 AND line < 543)) THEN
                            startr <= '1';  --active video, read pixel 4 clocks before end of blanking
                            ELSE    startr <= '0';
                                    startf <= '1';
                            END IF;

            WHEN OTHERS =>  startr <= '0';
                            startf <= '0';
            END CASE;
        END IF;
END PROCESS;

END fsmntsc;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE ntscpkg IS
COMPONENT ntsc PORT (
        clock,resete: IN std_logic;
        hsync,hblank: BUFFER std_logic;
        startr,startf: OUT std_logic);
END COMPONENT;
END ntscpkg;
```

A.2 Schematics

The board schematics, library files, and PC board layout for the real-time imager demonstration are located in `/homes/kfife` in ACCEL binary format.

```
/homes/kfife/Accel/Realtime/
        imager.sch
        imager.pcb
        format.sch
        format.pcb
        imager1.lib
```

Schematics for The Realtime demonstration are shown in Figures A.1 and A.2.

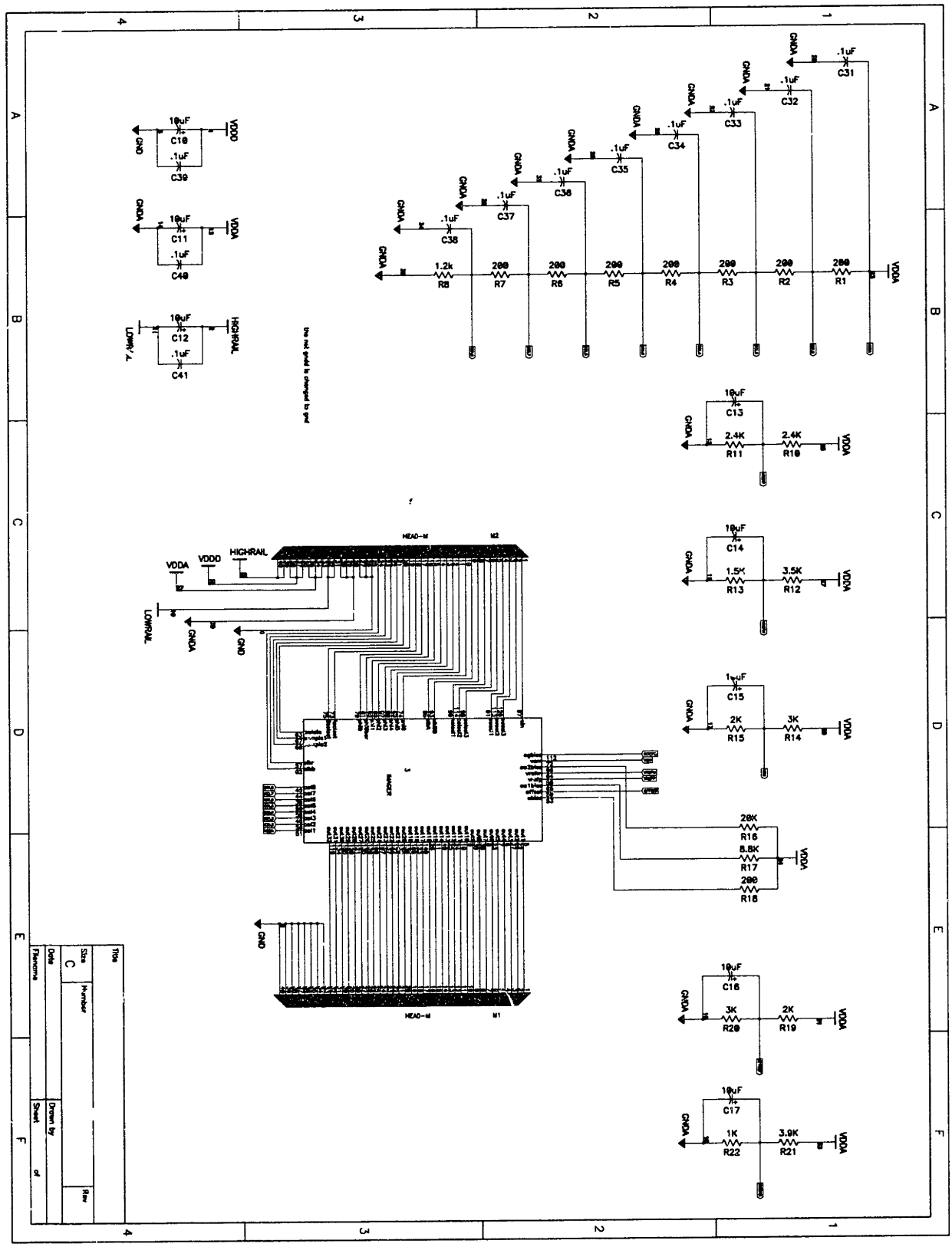

Figure A.1: Analog imager schematics.

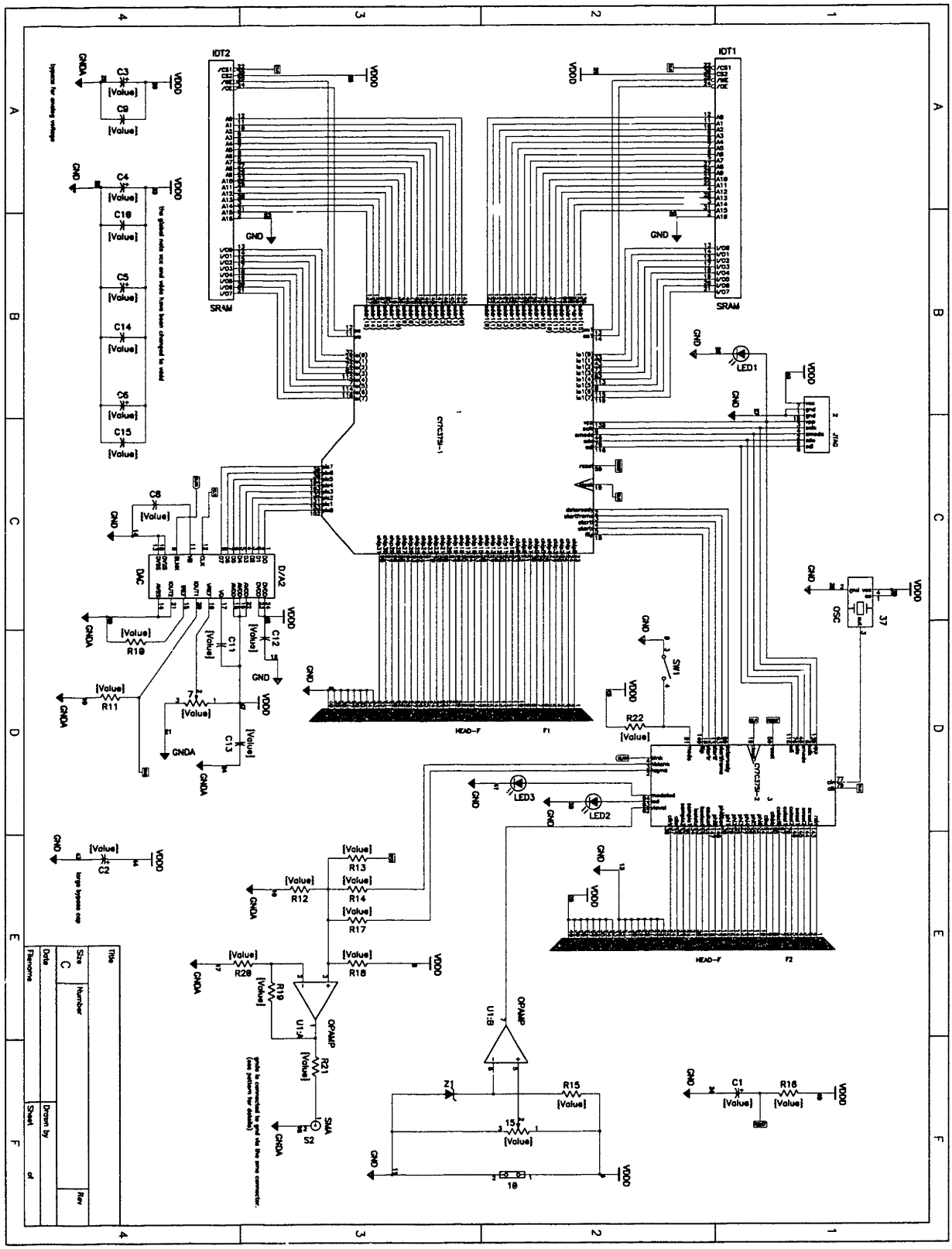

Figure A.2: Format converter schematics.

Appendix B

Automatic Brightness Adaptation Demonstration

B.1 VHDL

The VHDL files for the digital imager are located in:

```
/homes/kfife/Imager1/Stereo/
imsync.vhd
image.vhd
imagertiming.vhd

project name: imagerpc.pfg
```

The VHDL files used for frame averaging are located in:

```
/homes/kfife/Imager1/Stereo/ALU
topbox.vhd
boxcontrol.vhd
sample.vhd
serial2parallel.vhd
chipmux.vhd
startlatch.vhd
boxaverage.vhd

project name: boxaverage.pfg
```

The VHDL files used for targeting a wide dynamic range image are located in:

```
/homes/kfife/Imager1/Stereo/ALU
topwithlights.vhd
lights.vhd
```

```
topcomponents.vhd
gridlights.vhd
sendiris.vhd
compare.vhd
irisgen.vhd
sub.vhd
modegen.vhd
divider.vhd

project name: components.pfg
```

The VHDL files used for reformatting and displaying images are located in:

```
/homes/kfife/Imager1/Stereo/DATAPATH
monitordigital.vhd
serial2parallel.vhd
inout.vhd
sample.vhd
ntsc.vhd
flip.vhd
startlatch.vhd

project name: march999.pfg
```

The files used for the daughter card to the GuPPI card are located in:

```
/homes/kfife/Imager1/Stereo/GUPPI
topguppi.vhd

project name: guppi.pfg
```

B.1.1 Top Level - imsync.vhd

Location:

```
/homes/kfife/Imager1/Stereo/imsync.vhd
```

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity imsync IS PORT (
        clock,reset: IN std_logic;
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;
        Aselect, Bselect: BUFFER std_logic;
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
        modebd: IN std_logic;              --integration or linear mode
```

```
        colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
        phi5,phi6: OUT std_logic;
        ---
        manual: IN std_logic;  --says to use manual or autoiris
        cs: OUT std_logic;
        aclk: OUT std_logic;
        irisi: IN std_logic;
        ---

        chip: IN std_logic_vector(31 downto 0);
        outs: BUFFER std_logic;
        datacall: OUT std_logic;   --used to say when to start a frame and when data is ready
        ---
        irisbit: IN std_logic;  --used to transfer iris one bit at a time and sampled by clock

---POWER ON
        led: OUT std_logic;
---CLOCKS
        clk: OUT std_logic;
---MODE LED
        manled: OUT std_logic;
        modeled: OUT std_logic);
---

attribute pin_avoid of imsync:entity is "6 46 76 116";
-----------------------------------sclk,smode,SDO,SDI

attribute pin_numbers of imsync:entity is
"OUTS:2 " &

"PHI5:11 " &
"PHI6:14 " &

"LED:15 " &
"CLK:16 " &

"CLOCK:19 " &

"IRISBIT:22 " &

"PHI1:32 " &
"RSIN:33 " &
"MANLED:34 " &
"PHI3:42 " &
"MODELED:47 " &
"COLSEL(2):48 " &
"COLSEL(1):49 " &
"COLSEL(0):51 " &
"PHI2.52 " &
"PHI4:53 " &
"CLKB:65 " &
"SAMPLE1:68 " &
"ISOLATE:69 " &
"ASELECT:70 " &
"BSELECT.72 " &
"CLKA:73 " &
"CLKR:74 " &
"SAMPLE2:75 " &
"DATACALL:78 " &
"CLKBB.79 " &

"MANUAL:97 " &
"RESET:99 " &

"MODEBD:102 " &

"PHIOBAR:103 " &
"SWSEL1:104 " &
"SWSEL2:105 " &
"SWSEL3:106 " &

"CS.113 " &
"ACLK:114 " &
"IRISI:115 " &

"PHI0:119 " &

"CHIP(0):122 " &
"CHIP(1):123 " &
"CHIP(2):124 " &
"CHIP(3):125 " &
"CHIP(4):126 " &
"CHIP(5):127 " &
"CHIP(6):128 " &
"CHIP(7).129 " &
```

```
"CHIP(8):131 " &
"CHIP(9):132 " &
"CHIP(10):133 " &
"CHIP(11):134 " &
"CHIP(12):135 " &
"CHIP(13):136 " &
"CHIP(14):137 " &
"CHIP(15):138 " &
"CHIP(16):143 " &
"CHIP(17):144 " &
"CHIP(18):145 " &
"CHIP(19):146 " &
"CHIP(20):147 " &
"CHIP(21):148 " &
"CHIP(22):149 " &
"CHIP(23):150 " &
"CHIP(24):152 " &
"CHIP(25):153 " &
"CHIP(26):154 " &
"CHIP(27):155 " &
"CHIP(28):156 " &
"CHIP(29):157 " &
"CHIP(30):158 " &
"CHIP(31):159 ";
End imsync;

Use work.imagepkg.all;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF imsync IS

SIGNAL modebdnot: std_logic;
SIGNAL mode: std_logic;
SIGNAL reseti: std_logic;
SIGNAL resets: std_logic;

SIGNAL startframe: std_logic;
SIGNAL flash: std_logic_vector(4 downto 0);

BEGIN

image0: im PORT MAP(
        clock,resets,
        clkBB,clkA,swsel1,swsel2,swsel3,
        phi1,phi2,phi3,phi4,
        Aselect, Bselect,
        phi0,phi0bar,
        isolate,
        clkb,clkr,
        rsin,
        modebdnot,              --integration or linear mode
        colsel,
        sample1, sample2,
        phi5,phi6,
        ---
        manual,         --says to use manual or autoiris
        cs,
        aclk,
        irisi,          --signal from ADC
        ---
        chip,
        mode,
        reseti,

        startframe,

        outs,
        datacall,       --used to say when to start a frame and when data is ready
        ---
        irisbit);       --used to transfer iris one bit at a time and sampled by clock

modebdnot <= not modebd;

led <= flash(4) or irisbit;

modeled <= '1' when mode = '1' else '0';

manled <= '1' when manual = '1' else '0';

resets <= reset when manual = '1' else reseti,

-----------
clk <= clock; --slightly delayed clock
-----------
```

```
process begin
wait until clock = '1';
if startframe = '1' then
flash <= flash + 1;
end if;
end process;

END behavior;
```

B.1.2 Data Protocol and Barrier Generation - image.vhd

Location:

/homes/kfife/Imager1/Stereo/image.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity im IS PORT (
        clock,reset: IN std_logic;                      --reset active low
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;          --ADC timing
        Aselect, Bselect: BUFFER std_logic;             --ADC selector for pad driver
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
        modebd: IN std_logic;                                   --integration or linear mode
        colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
        phi5,phi6:OUT std_logic;
        ---
        manual: IN std_logic;  --says to use manual or autoiris
        cs: OUT std_logic;
        aclk: OUT std_logic;
        irisi: IN std_logic;
        ---
        chip: IN std_logic_vector(31 downto 0);
--      chip: BUFFER std_logic_vector(31 downto 0);


        mode: BUFFER std_logic;   -- ported mode so that this can be used in package
        reseti: OUT std_logic;  --ported for internal signal when packaged

        startframe:     BUFFER std_logic;

        outs: BUFFER std_logic;
        datacall: OUT std_logic;
        irisbit: IN std_logic);  --used to transfer iris/mode/reset

End im;

--16384 32-bit outputs per frame  At 24.576MHz -> 409600 clks per frame, so / by 25,
--hence i = 0 to 24.

--As of 1/3/99 I am using a 49 MHz clk so I divide by 50 and send the output at 49MHz.
--Sending data for the 32 intervals and then wait for 18

USE WORK.std_arith.ALL,
USE WORK.imagertimingpkg.all; -- all the signal for the chip except for colsel and 32 outs


ARCHITECTURE fsmim OF im IS

------------------
        SIGNAL count. std_logic_vector(5 downto 0);
        SIGNAL i  integer range 0 to 49;              --i is used to divide the clock by 50.
        SIGNAL r: integer range 0 to 255;             --r is the number of rows.
------------------

        SIGNAL lock: std_logic;

        SIGNAL A. std_logic_vector(7 downto 0), --used in comparator
        SIGNAL compare. boolean;        --output of comparator
        Attribute synthesis_off of compare: SIGNAL is TRUE;

        SIGNAL iriscount  std_logic_vector(2 downto 0),

        SIGNAL iris  std_logic_vector(7 downto 0),

        SIGNAL row  std_logic_vector(1 downto 0),
```

```
        SIGNAL irisin: std_logic_vector(7 downto 0);  --holding place for adc iris
        SIGNAL irisbd: std_logic_vector(7 downto 0);  --holding place for adc iris registered
        SIGNAL lock1: std_logic;
        SIGNAL csadc: std_logic;
        SIGNAL getbit: std_logic;

--      SIGNAL startframe: std_logic;
        SIGNAL dataready: std_logic;   --used to say when to start a frame and when data is ready

        TYPE states IS (idle,check,wait1,wait2,modestate,bits);
        SIGNAL state: states;

        SIGNAL modeobd: std_logic;
        SIGNAL irisobd: std_logic_vector(7 downto 0);
        SIGNAL count16: std_logic_vector(3 downto 0);
        SIGNAL sample: boolean;
        SIGNAL sampleadc: boolean;

BEGIN


--standard imagertiming--
imagertiming0: imagertiming port map (
        clock,reset,                                    --reset active low
        clkBB,clkA,swsel1,swsel2,swsel3,
        phi1,phi2,phi3,phi4,                            --ADC timing
        Aselect, Bselect,                               --ADC selector for pad driver
        phi0,phi0bar,
        isolate,
        clkb,clkr,
        rsin,
        sample1, sample2,
        phi5,phi6,                              --used to say when to start a frame and when data is ready
        count,
        i,                                      --i is used to divide the clock by 50.
        r);                                     --r is the row count
---------------------------

--decoding irisbit--------------
Autoiris: PROCESS (clock,reset,startframe,irisbit)

begin

If reset = '0' then
state <= idle;

ELSIF clock'EVENT AND clock = '1' THEN

                CASE state IS
                        WHEN idle =>    iriscount <= "000";
                                        IF startframe = '1' THEN
                                        state <= check;
                                        ELSE state <= idle;
                                        END IF;
                        WHEN check =>   IF irisbit = '0' THEN
                                        state <= wait1;
                                        ELSE state <= idle;  --error
                                        END IF;
                        WHEN wait1 =>   IF irisbit = '1' THEN
                                        state <= wait2;
                                        ELSE state <= wait1;
                                        END IF;
                        WHEN wait2 =>   IF irisbit = '0' THEN
                                        state <= bits;
                                        ELSE state <= wait2;
                                        END IF;
                        WHEN bits => irisobd <= irisobd(6 downto 0) & irisbit;   --send MSB first
                                        iriscount <= iriscount + 1;

                                        IF iriscount = "111" THEN
                                        state <= modestate;
                                        ELSE state <= bits;
                                        END IF;
                        WHEN modestate => modeobd <= irisbit;  --9th bit of data is the mode.
                                        state <= idle;
                        WHEN OTHERS =>  state <= idle;
                END CASE;

END IF;

END PROCESS Autoiris;
-------------------------------------------------------------------------------


--reset signal generated from irisbit held high for 16 counts-----------------
Resetsignal: PROCESS (clock,irisbit,count16)
```

```
begin

IF clock'EVENT AND clock = '1' THEN

                        IF irisbit = '1' and (count16 /= "1111") THEN
                        count16 <= count16 + 1;
                        ELSIF (irisbit = '0') THEN count16 <= "0000";
                        END IF;

                        IF (count16 = "1111") THEN
                        reseti <= '0';
                        ELSE reseti <= '1';
                        END IF;
END IF;
END PROCESS Resetsignal;
-------------------------------------------------------------------------------

process begin
wait until clock = '1';  --outs gets registered to reduce the propagation delay.

IF i = 17 THEN sample <= true; END IF;
IF i = 49 THEN sample <= false; END IF;

case i is
        when 17 =>      outs <= chip(16);
        when 18=>       outs <= chip(20);
        when 19=>       outs <= chip(24);
        when 20=>       outs <= chip(28);

        when 21 =>      outs <= chip(15);
        when 22 =>      outs <= chip(11);
        when 23 =>      outs <= chip(7);
        when 24 =>      outs <= chip(3);

        when 25 =>      outs <= chip(17);
        when 26 =>      outs <= chip(21);
        when 27 =>      outs <= chip(25);
        when 28 =>      outs <= chip(29);

        when 29 =>      outs <= chip(14);
        when 30 =>      outs <= chip(10);
        when 31 =>      outs <= chip(6);
        when 32 =>      outs <= chip(2);

        when 33 =>      outs <= chip(18);
        when 34 =>      outs <= chip(22);
        when 35 =>      outs <= chip(26);
        when 36 =>      outs <= chip(30);

        when 37 =>      outs <= chip(13);
        when 38 =>      outs <= chip(9);
        when 39 =>      outs <= chip(5);
        when 40 =>      outs <= chip(1);

        when 41 =>      outs <= chip(19);
        when 42 =>      outs <= chip(23);
        when 43 =>      outs <= chip(27);
        when 44 =>      outs <= chip(31);

        when 45 =>      outs <= chip(12);
        when 46 =>      outs <= chip(8);
        when 47 =>      outs <= chip(4);
        when others =>  outs <= chip(0);

END case;


end process;
-------datacall---startframe---startframelow--------------------------
process(clock,i,r,lock) begin                   --registered to prevent glitches
if rising_edge(clock) then
        ---startframe
        IF ((r = 0) AND (i = 4) AND (lock = '0')) THEN
                startframe <= '1';
                lock <= '1';
        else startframe <= '0';
        END IF;
        IF r = 1 THEN lock <= '0'; END IF;
        -------------
end if;
end process;

dataready <= (not clock) when sample else '0'; --sample data on not clock

datacall <= dataready or startframe;  --combines startframe and dataready into one.
-------------------------------------------------------------------
```

```
---------------------------------barrier function
--process(r,mode,clock,colsel)
--begin
--
--IF clock'EVENT AND clock = '1' THEN
--
--if mode = '1' then     --integration mode; approximating log function
--      case r is
--              when 255 => colsel <= "000"; --highest potential(reset = 5v)
--              when 0   => colsel <= "001";
--              when 127 => colsel <= "010";
--              when 191 => colsel <= "011";
--              when 223 => colsel <= "100";
--              when 239 => colsel <= "101";
--              when 247 => colsel <= "110";
--              when 251 => colsel <= "111"; --lowest potential( = 2.5v)
--              when others => colsel <= colsel;
--      end case;
--else
--      case r is      --linear mode
--              when 255 => colsel <= "000";
--              when 0   => colsel <= "111";
--              when others => colsel <= colsel;
--      end case;
--
--end if;
--
--END IF;
--
--end process;
-----------------------------------------------------------------

------------mux for selecting iris; computation of log function.
with colsel select
        A <=             iris when "000",
                        '1' & iris(7 downto 1) when "001",
                        "11" & iris(7 downto 2) when "010",
                        "111" & iris(7 downto 3) when "011",
                        "1111" & iris(7 downto 4) when "100",
                        "11111" & iris(7 downto 5) when "101",
                        "111111" & iris(7 downto 6) when "110",
                        "11111111"    when "111",
                        "00000000" when others;
-----------------------------------------------------------------

---8-b comparator-------------------------------------------------
compare <= r = to_integer(A); --compares row to iris to determine when to step barrier
-----------------------------------------------------------------
-----------------------------------------------------------------
Barrier:PROCESS(clock,r)

BEGIN

IF r = 255
then colsel <= "000";

ELSIF clock'EVENT AND clock = '1' THEN

        IF mode = '1' THEN
                IF r = 254 THEN
                colsel <= "111";  --at least grab the last little step
                ELSIF compare THEN colsel <= colsel + 1;
                END IF;
        else
                IF compare THEN colsel <= "111";
                END IF;
        END IF;

END IF;
END PROCESS Barrier;
-----------------------------------------------------------------

--------------getting iris from adc-----------------------------------
Getiris: Process(startframe,clock,row,lock1,getbit,irisi) --get iris from adc

begin

IF clock'EVENT AND clock = '1' THEN

        IF (manual = '1') THEN
                IF sampleadc and (lock1 = '0') THEN
                irisin <= irisin(6 downto 0) & irisi;  --send MSB first
                lock1 <= '1';
                END IF;
```

```
                IF not sampleadc THEN           --lock is set at the beginning of rows
                lock1 <= '0';
                END IF;

                IF (r = 0) THEN
                irisbd <= irisin;
                END IF;
        END IF;

END IF;
END PROCESS Getiris;

--timing for serial adc--
with r select
        getbit <= '1' when 5|7|9|11|13|15|17|19,
                '0' when others;

with r select
        sampleadc <= true when 6|8|10|12|14|16|18|20,
                false when others;

with r select
        csadc <= '0' when 0 to 19,
                '1' when others;

row <= to_std_logic_vector(r,2);        --converts r to std_logic_vector
aclk <= row(0);                         --takes the lsb of row for a2d clock


cs <= csadc when (manual = '1') else '1';
mode <= modebd when (manual = '1') else modeobd;
iris <= irisbd when (manual = '1') else irisobd;
-------------------------------------------------------------------------

END fsmim;
----

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE imagepkg IS
COMPONENT im PORT (
        clock,reset: IN std_logic;                      --reset active low
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;          --ADC timing
        Aselect, Bselect: BUFFER std_logic,             --ADC selector for pad driver
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic,
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
        modebd: IN std_logic;                                   --integration or linear mode
        colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic,
        phi5,phi6:OUT std_logic;
        ---
        manual: IN std_logic;  --says to use manual or autoiris
        cs: OUT std_logic;
        aclk: OUT std_logic;
        irisi: IN std_logic;
        ---
        chip: IN std_logic_vector(31 downto 0);

        mode: BUFFER std_logic;   -- ported mode so that this can be used in package
        reseti: OUT std_logic;  --ported for internal signal when packaged

        startframe:     BUFFER std_logic;

        outs: BUFFER std_logic;
        datacall: OUT std_logic;
        irisbit. IN std_logic);  --used to transfer iris/mode/reset
END COMPONENT;
END imagepkg;
```

B.1.3 Package for all Digital Imager Signals - imagertiming.vhd

Location:

/homes/kfife/Imager1/Stereo/imagertiming.vhd

```
--All signals for the imager chip except for colsel and the 32 bit outputs.
Library ieee;
Use ieee.std_logic_1164.all;
Entity imagertiming IS PORT (
        clock,reset: IN std_logic;                      --reset active low
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;          --ADC timing
        Aselect, Bselect: BUFFER std_logic;             --ADC selector for pad driver
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
--      colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
        phi5,phi6:OUT std_logic;                --used to say when to start a frame and when data is ready
        count: BUFFER std_logic_vector(5 downto 0);
        i: BUFFER integer range 0 to 49;            --i is used to divide the  clock by 50.
        r: BUFFER integer range 0 to 255);     --r is the row count
End imagertiming;

USE WORK.std_arith.ALL;


ARCHITECTURE behavior OF imagertiming IS

        SUBTYPE v3 is std_logic_vector(2 downto 0);
        alias switch: v3 is count(2 downto 0);

        SIGNAL phih1,phih2,phih3,phih4: boolean;

BEGIN

PROCESS (clock,i,switch,reset)

BEGIN

        IF      reset = '0' THEN
                i <= 0;
                r <= 0;
                count <= "000000";

        ELSIF clock'EVENT AND clock = '1' THEN
                if i = 49 then              --divide down the system clock
                        i <= 0;

                        --phi1-4
                        if (switch = "111") or (switch = "000")then
                        phih3 <= TRUE; else phih3 <= FALSE; end if;

                        if (switch = "001") or (switch = "010")then
                        phih4 <= TRUE; else phih4 <= FALSE; end if;

                        if (switch = "011") or (switch = "100")then
                        phih1 <= TRUE; else phih1 <= FALSE; end if;

                        if (switch = "101") or (switch = "110")then
                        phih2 <= TRUE; else phih2 <= FALSE; end if;

                        --Aselect and Bselect
                        if ( count = "000000") then
                        Bselect <= '1'; end if;
                        if ( count = "011111") then
                        Bselect <= '0'; end if;
                        if ( count = "100000") then
                        Aselect <= '1';  end if;
                        if ( count = "111111") then
                        Aselect <= '0'; end if;

                        --sample1 and sample2 and isolate
                        if ( count = "100000") then
                        sample2 <= '0'; end if;
                        if ( count = "100001") then
                        sample1 <= '1'; end if;
                        if ( count = "101101") then
                        sample1 <= '0';  end if;
                        if ( count = "101110") then
                        sample2 <= '1'; end if;

                        if ( count = "100001") then
                        isolate <= '1'; end if;
                        if ( count = "111011") then
                        isolate <= '0'; end if;

                        --clkb
                        if ( count = "101101") then
                        clkb <= '1';
```

```
                        else clkb <= '0';
                        end if;

                        --clkr
                        if ( count = "100000") then
                        clkr <= '1';
                        else clkr <= '0';
                        end if;

                        if count = "111111" then
                                count <= "000000";
                                if r = 255 then          --counting rows
                                        r <= 0;
                                else r <= r + 1;
                                end if;
                        else count <= count + 1;
                        end if;

                else i <= i + 1;
                end if;

        END IF;
END PROCESS;

------------non-overlapping phis----------
phi1 <= '1' when phih1 and not phih4 else '0';
phi2 <= '1' when phih2 and not phih1 else '0';
phi3 <= '1' when phih3 and not phih2 else '0';
phi4 <= '1' when phih4 and not phih3 else '0';
-----------------------------------------

clkA <= '0' when switch = "000" else '1';
clkBB <= '0' when switch = "100" else '1';

phi0 <= '1' when (count(4 downto 0) > "11011") else '0';
phi0bar <= not phi0;


--rsin
rsin <= '1' when r = 255 else '0';

--phi5, phi6
phi5 <= Aselect;
phi6 <= Bselect;

swsel2 <= switch(0);
swsel1 <= switch(1);
swsel3 <= switch(2);

END behavior;

Library ieee;
Use ieee.std_logic_1164.all,
PACKAGE imagertimingpkg IS
COMPONENT imagertiming PORT (
        clock,reset: IN std_logic;                      --reset active low
        clkBB,clkA,swsel1,swsel2,swsel3: OUT std_logic;
        phi1,phi2,phi3,phi4: BUFFER std_logic;          --ADC timing
        Aselect, Bselect: BUFFER std_logic;             --ADC selector for pad driver
        phi0,phi0bar: BUFFER std_logic;
        isolate: BUFFER std_logic;
        clkb,clkr: BUFFER std_logic;
        rsin: BUFFER std_logic;
--      colsel: BUFFER std_logic_vector(2 downto 0);
        sample1, sample2: BUFFER std_logic;
        phi5,phi6:OUT std_logic;                --used to say when to start a frame and when data is ready
        count: BUFFER std_logic_vector(5 downto 0);
        i: BUFFER integer range 0 to 49;            --i is used to divide the  clock by 50.
        r: BUFFER integer range 0 to 255);     --r is the row count
END COMPONENT;
END imagertimingpkg;
```

B.1.4 Top Level - topbox.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/topbox.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity topbox IS PORT (
        clk:            IN std_logic;
        clk49:          IN std_logic;
        clkout:         OUT std_logic;
        clock:          IN std_logic;
        clockout:       BUFFER std_logic; --This is the half clock.
        outs:           IN std_logic;
        datacall:       IN std_logic;
        index:          OUT std_logic_vector(3 downto 0);
        CLR:            OUT std_logic;
        LDbot:          OUT std_logic;
        avg:            OUT std_logic_vector(7 downto 0);
        LDall:          OUT std_logic;
        interin:        IN std_logic_vector(3 downto 0); --in from monitor digital
        interout:       OUT std_logic_vector(3 downto 0)); --readable to gridlights

attribute pin_avoid of topbox:entity is "6 46 76 116";
-----------------------------------sclk,smode,SDO,SDI

attribute pin_numbers of topbox:entity is
"AVG(0):2 " &
"AVG(1):9 " &
"DATACALL:19 " &
"CLK:22 " &
"AVG(2):23 " &
"AVG(3):27 " &
"AVG(4):30 " &
"CLOCKOUT:33 " &
"OUTS:59 " &
"AVG(5):64 " &
"CLOCK:99 " &
"INDEX(0):122 " &
"INDEX(1):123 " &
"INDEX(2):124 " &
"INDEX(3):125 " &
"CLR:131 " &
"LDBOT:132 " &
"LDALL:133 " &
"INTERIN(0):135 " &
"INTERIN(1):136 " &
"INTERIN(2):137 " &
"INTERIN(3):138 " &
"clk49:143 " &
"clkout:144 " &
"AVG(6):152 " &
"AVG(7):153 " &
"INTEROUT(0):155 " &
"INTERGUT(1):156 " &
"INTEROUT(2):157 " &
"INTEROUT(3):158 ";

End topbox;

USE WORK.std_arith.ALL;

USE WORK.serial2parallelpkg.all;
USE WORK.startlatchpkg.all;
USE WORK.samplepkg.all;
USE WORK.chipmuxpkg.all;
USE WORK.boxcontrolpkg.all;
USE WORK.boxaveragepkg.all;

ARCHITECTURE behavior OF topbox IS
SUBTYPE v1 is std_logic;
SUBTYPE v2 is std_logic_vector(1 downto 0);
SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);
SUBTYPE v8 is std_logic_vector(7 downto 0);
SUBTYPE v32 is std_logic_vector(31 downto 0);

SIGNAL datain: v32;
SIGNAL chip: v32;
SIGNAL dav: std_logic;
SIGNAL rdav: std_logic;
SIGNAL samplenow: std_logic;
SIGNAL start: std_logic;
SIGNAL startreset: std_logic;
SIGNAL done: std_logic;
SIGNAL clearbits: boolean;
SIGNAL Readin: std_logic_vector(2 downto 0);
SIGNAL  halfbyte: v4;
SIGNAL  column: v2;
SIGNAL  row: v2;
SIGNAL  newpixel: boolean;
```

```
SIGNAL  newbitplane: boolean;
SIGNAL  newblock: boolean;

Attribute synthesis_off of halfbyte: SIGNAL is TRUE;

BEGIN

serial2parallel0: serial2parallel PORT MAP(
        clk=>clk,
        start=>start,
        data=>datain,
        outs=>outs,
        dav=>dav,
        rdav=>rdav,
        datacall=>datacall);

startlatch0: startlatch PORT MAP(
        clock=>clock,
        start=>start,
        done=>done,
        startQ=>startreset);

sample0: sample PORT MAP(
        clock=>clock,
        dav=>dav,
        rdav=>rdav,
        samplenow=>samplenow,
        datain=>datain,
        dataout=>chip);

chipmux0: chipmux PORT MAP(
        chip=>chip,
        output=>halfbyte,
        clearbits=>clearbits,
        Readin=>Readin);

boxcontrol0: boxcontrol PORT MAP(
        clock=>clock,
        startreset=>startreset,
        samplenow=>samplenow,
        Readin=>Readin,
        clearbits=>clearbits,
        done=>done,
        column=>column,
        row=>row,
        newpixel=>newpixel,
        newbitplane=>newbitplane,
        newblock=>newblock,
        LDbot=>LDbot,
        CLR=>CLR,
        LDall=>LDall);

boxaverage0: boxaverage PORT MAP(
        clock=>clock,
        input=>halfbyte,
        column=>column,
        newpixel=>newpixel,
        newbitplane=>newbitplane,
        newblock=>newblock,
        avg=>avg);

--This is the index for gridlights
index <= row & column;
-----

process begin
WAIT UNTIL (clk = '1');
clockout <= not clockout;               --divide clk by two
end process;

interout <= interin;
clkout<=clk49;

END behavior;
```

B.1.5 Controller for Arithmetic Functions - boxcontrol.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/boxcontrol.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity boxcontrol IS PORT (
        clock:          IN std_logic;
        startreset: IN std_logic;               --this comes from the Q output of an rs latch, done resets it.
        samplenow:      IN std_logic;
        Readin:         BUFFER std_logic_vector(2 downto 0);
        clearbits:      OUT boolean;
        done:           OUT std_logic;          --reset for the startlatch
        column:         BUFFER std_logic_vector(1 downto 0);
        row:            BUFFER std_logic_vector(1 downto 0);
        newpixel:       OUT boolean;            --for boxaverage
        newbitplane:    OUT boolean;            --for boxaverage
        newblock:       OUT boolean;            --for boxaverage
        LDbot:          OUT std_logic;          --for compare
        CLR:            OUT std_logic;          --for compare
        LDall:          OUT std_logic);         --signal to Load and clear the frame average --also signal to start division
End boxcontrol;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF boxcontrol IS

SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);

TYPE states IS (doread,loadaverage,waitread,settleavg);
SIGNAL state: states;

SIGNAL countin: std_logic_vector(14 downto 0);
SIGNAL columnindex: std_logic_vector(1 downto 0);

BEGIN

------

Readin <= countin(2 downto 0); --8 sets of 4-bit wide column-separated packets in one 32 bit sample
clearbits <= FALSE when state = doread else TRUE;

------
PROCESS (state,clock,startreset,Readin,countin)

BEGIN


IF clock'EVENT AND clock = '1' THEN

        CASE state IS
                        WHEN doread =>
                                        IF (countin = "111111111111111") THEN
                                                state <= settleavg;
                                        ELSIF (Readin = "111") then
                                                state <= waitread;
                                        END IF;

                                        countin <= countin + 1;

                        WHEN settleavg => state <= loadaverage;
                                        IF (row = "00") THEN CLR <= '1'; END IF; --this loads in avg0 in both bright and dark registers

                        WHEN loadaverage =>
                                        IF (columnindex = "11") then
                                        state <= waitread;
                                        row <= row + 1;
                                        END IF;
                                        CLR <= '0';
                                        columnindex <= columnindex + 1;

                        WHEN  waitread =>
                                        IF (samplenow = '1') then
                                        state <= doread;
                                        END IF;

                                        IF (startreset = '1') THEN
                                        countin <= (others => '0');
                                        columnindex <= "00";
```

```
                                        row <= "00";
                                        done <= '1';
                                        LDall <= '1';
                                        ELSE
                                        done <= '0';
                                        LDall <= '0';
                                        END IF;
                        WHEN OTHERS => state <= waitread;

        END CASE;

END IF;
END PROCESS;
-------

LDbot <= '1' when (state = loadaverage) else '0';
column <= columnindex;

-------booleans for boxaverage-----
process begin
wait until clock = '1';

        IF (countin(7 downto 0) = "11111111") THEN
        newpixel <= TRUE;
        ELSIF (countin(4 downto 0) = "11111") THEN
        newbitplane <= TRUE;
        ELSE
        newpixel <= FALSE;
        newbitplane <= FALSE;
        END IF;

        IF(columnindex = "11") THEN
        newblock <= TRUE;
        ELSE
        newblock <= FALSE;
        END IF;

end process;
-----------------------------------

END behavior;


Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE boxcontrolpkg IS
COMPONENT boxcontrol PORT (
        clock:          IN std_logic;
        startreset: IN std_logic;               --this comes from the Q output of an rs latch, done resets it.
        samplenow:      IN std_logic;
        Readin:         BUFFER std_logic_vector(2 downto 0);
        clearbits:      OUT boolean;
        done:           OUT std_logic;          --reset for the startlatch
        column:         BUFFER std_logic_vector(1 downto 0);
        row:            BUFFER std_logic_vector(1 downto 0);
        newpixel:       OUT boolean;            --for boxaverage
        newbitplane:OUT boolean;                --for boxaverage
        newblock:       OUT boolean;            --for boxaverage
        LDbot:          OUT std_logic;          --for compare
        CLR:            OUT std_logic;          --for compare
        LDall:          OUT std_logic);         --signal to Load and clear the frame average --also signal to start division
END COMPONENT;
END boxcontrolpkg;
```

B.1.6 Sychronous Sample - sample.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/sample.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity sample IS PORT (
        clock:          IN std_logic;
        dav:            IN std_logic;
        rdav:           OUT std_logic;
        samplenow:      OUT std_logic;
        datain:         IN std_logic_vector(31 downto 0);
        dataout:        OUT std_logic_vector(31 downto 0));
End sample;
```

```
USE WORK.std_arith ALL;

ARCHITECTURE behavior OF sample IS

BEGIN

PROCESS(clock,dav,datain)

BEGIN

IF clock'EVENT AND clock = '1' THEN

               IF dav = '1' THEN
               dataout <= datain;
               samplenow <= '1';
               rdav <= '1';
               ELSE
               samplenow <= '0';
               rdav <= '0';
               END IF;
END IF;

END PROCESS;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE samplepkg IS
COMPONENT sample PORT (
        clock:          IN std_logic;
        dav:            IN std_logic;
        rdav:           OUT std_logic;
        samplenow:      OUT std_logic;
        datain:         IN std_logic_vector(31 downto 0);
        dataout:        OUT std_logic_vector(31 downto 0));
END COMPONENT;
END samplepkg;
```

B.1.7 Serial to Parallel Converter - serial2parallel.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/serial2parallel.vhd

```
-------------------------------------------------------------------
--data.vhd uses datacall from imager to create a 32 bit word from
--the 1-bit input 'outs' from imager
-------------------------------------------------------------------
Library ieee;
Use ieee.std_logic_1164.all;
Entity serial2parallel IS PORT (
        clk:            IN std_logic;           --clk on board receiving end
        start:          BUFFER std_logic;       --start signal for the start of frame
        data:           BUFFER std_logic_vector(31 downto 0);
        outs:           IN std_logic;           --bit input from imager
        dav:            OUT std_logic;          --indication of when data is ready
        rdav:           IN std_logic;           --resets dav asynchronously
        datacall:       IN std_logic);          --clock for outs start signal embedded
End serial2parallel;

USE WORK.std_arith.ALL;

ARCHITECTURE datapath OF serial2parallel IS

SIGNAL count: std_logic_vector(4 downto 0);     --clocked with datacall
SIGNAL ckcount: std_logic_vector(2 downto 0);   --counter for check for deadzone in datacall
SIGNAL check: std_logic_vector(4 downto 0);     --used to sample count value
SIGNAL previouscheck: std_logic_vector(4 downto 0);     --used to sample check value

SIGNAL breakcondition: boolean;
SIGNAL idlecondition: boolean;
SIGNAL normalcondition: boolean;

SIGNAL resetcount: std_logic;
TYPE states IS (normal,break,waitsample,idle);
SIGNAL state : states;

BEGIN
```

```
--------serial to parallel conversion-----------
PROCESS(datacall,rdav,resetcount,start)
begin

IF rdav = '1' THEN
dav <= '0';

ELSIF (resetcount = '1' or start = '1') THEN
count <= (others => '0');

ELSIF datacall'EVENT AND datacall = '1' THEN

        data <= data(30 downto 0) & outs;
        count <= count + 1;

        IF (count = "11111") THEN
        dav <= '1';
        END IF;

END IF;
END PROCESS;
-----------------------------------------------

---decoding the start frame signal-------------
--The process samples the count value with check. Check is sampled by previouscheck.
--By comparing the two values, the process determines when there is a break in the
--datacall clock. It then looks for the start signal by probing check and previouscheck
--during a 4 period clock sequence. If found, the start signal is fired. Otherwise the
--process resumes normal behavior.

breakcondition <= (ckcount = "011") and (previouscheck = check);
idlecondition <= (ckcount = "111") and (previouscheck = check);
normalcondition <= (check(0) = '1');

process(clk)
begin

IF rising_edge(clk) THEN
        check <= count;
        previouscheck <= check;
        IF (previouscheck = check) THEN
        ckcount <= ckcount + 1;
        ELSE
        ckcount <= (others => '0');
        END IF;

        CASE state is
                        when normal =>  start <= '0';
                                                resetcount <= '0';
                                        IF breakcondition THEN
                                                state <= break;
                                        END IF;

                        when break =>
                                        IF idlecondition THEN
                                                resetcount <= '1';
                                                state <= waitsample;
                                        ELSIF (previouscheck /= check) THEN
                                                start <= '1';
                                                state <= waitsample;
                                        END IF;

                        when waitsample =>      resetcount <= '0';
                                                        start <= '0';
                                                        state <= idle;

                        when idle => start <= '0';
                                        IF normalcondition THEN
                                                state <= normal;
                                        END IF;

                        when others => state <= normal;
        END CASE;

END IF;
end process;
---------------------------------------

END datapath;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE serial2parallelpkg IS
COMPONENT serial2parallel PORT (
        clk:            IN std_logic;           --clk on board receiving end
```

```
        start:          BUFFER std_logic;       --start signal for the start of frame
        data:           BUFFER std_logic_vector(31 downto 0);
        outs:           IN std_logic;           --bit input from imager
        dav:            OUT std_logic;          --indication of when data is ready
        rdav:           IN std_logic;           --resets dav asynchronously
        datacall:       IN std_logic);          --clock for outs start signal embedded
END COMPONENT;
END serial2parallelpkg;
```

B.1.8 Selector - chipmux.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/chipmux.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity chipmux IS PORT (
        chip:           IN std_logic_vector(31 downto 0);
        output:         OUT std_logic_vector(3 downto 0);
        clearbits:      IN boolean;
        Readin:         IN std_logic_vector(2 downto 0));
End chipmux;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF chipmux IS

SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);

SIGNAL outputprime: v4;

-------******Imager output to PLD input configuration******-------
        alias chipA: v4 is chip(31 downto 28);
        alias chipB: v4 is chip(27 downto 24);
        alias chipC: v4 is chip(23 downto 20);
        alias chipD: v4 is chip(19 downto 16);
        alias chipE: v4 is chip(15 downto 12);
        alias chipF: v4 is chip(11 downto 8);
        alias chipG: v4 is chip(7 downto 4);
        alias chipH: v4 is chip(3 downto 0);
--each of these correspond to 1 bit for each of 4 columns
-----------------------------------------------------------------

BEGIN

------
PROCESS BEGIN
        CASE Readin is
                WHEN "000" => outputprime <= chipA;
                WHEN "001" => outputprime <= chipB;
                WHEN "010" => outputprime <= chipC;
                WHEN "011" => outputprime <= chipD;
                WHEN "100" => outputprime <= chipE;
                WHEN "101" => outputprime <= chipF;
                WHEN "110" => outputprime <= chipG;
                WHEN others => outputprime <= chipH;
        END CASE;
END PROCESS;
------

output <= "0000" when clearbits else outputprime;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE chipmuxpkg IS
COMPONENT chipmux PORT (
        chip:           IN std_logic_vector(31 downto 0);
        output:         OUT std_logic_vector(3 downto 0);
        clearbits:      IN boolean;
        Readin:         IN std_logic_vector(2 downto 0));
END COMPONENT;
END chipmuxpkg;
```

B.1.9 Start of Frame - startlatch.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/startlatch.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity startlatch IS PORT (
        clock:          IN std_logic;
        start:          IN std_logic;
        done:           IN std_logic;
        startQ:         BUFFER std_logic);
End startlatch;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF startlatch IS

        SIGNAL startsync: std_logic;
BEGIN

--Q <= not(reset or not(set or Q));  --rs latch

startQ <= not(done or not(start or startQ));

process begin
wait until clock = '1';
startsync <= start;
end process;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE startlatchpkg IS
COMPONENT startlatch PORT (
        clock:          IN std_logic;
        start:          IN std_logic;
        done:           IN std_logic;
        startQ:                 BUFFER std_logic);
END COMPONENT;
END startlatchpkg;
```

B.1.10 Generating Averages - boxaverage.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/boxaverage.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity boxaverage IS PORT (
        clock:          IN std_logic;
        input:          IN std_logic_vector(3 downto 0);
        column:         IN std_logic_vector(1 downto 0);
        newpixel:       IN boolean;
        newbitplane:  IN boolean;
        newblock:       IN boolean;
        avg:            OUT std_logic_vector(7 downto 0));
End boxaverage;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF boxaverage IS

SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);

SUBTYPE v21 is std_logic_vector(20 downto 0);

SIGNAL accum0: v21;
SIGNAL accum1: v21;
SIGNAL accum2: v21;
SIGNAL accum3: v21;
```

```
SUBTYPE v8 is std_logic_vector(7 downto 0);

SIGNAL  avg0: v8;
SIGNAL  avg1: v8;
SIGNAL  avg2: v8;
SIGNAL  avg3: v8;

BEGIN

avg0 <= accum0(12 downto 5) when (accum0(13) = '0') else x"FF";
avg1 <= accum1(12 downto 5) when (accum1(13) = '0') else x"FF";
avg2 <= accum2(12 downto 5) when (accum2(13) = '0') else x"FF";
avg3 <= accum3(12 downto 5) when (accum3(13) = '0') else x"FF";
----
--The MSBs are send first
----
Process begin
wait until (clock = '1');

IF (input(0) = '1') THEN
Accum0 <= Accum0 + 1;
ELSIF (newblock) THEN
Accum0 <= (others => '0');
ELSIF (newpixel) THEN
Accum0 <= Accum0(6 downto 0) & Accum0(20 downto 7);
ELSIF (newbitplane) THEN
Accum0 <= Accum0(19 downto 0) & Accum0(20);
END IF;


IF (input(1) = '1') THEN
Accum1 <= Accum1 + 1;
ELSIF (newblock) THEN
Accum1 <= (others => '0');
ELSIF (newpixel) THEN
Accum1 <= Accum1(6 downto 0) & Accum1(20 downto 7);
ELSIF (newbitplane) THEN
Accum1 <= Accum1(19 downto 0) & Accum1(20);
END IF;


IF (input(2) = '1') THEN
Accum2 <= Accum2 + 1;
ELSIF (newblock) THEN
Accum2 <= (others => '0');
ELSIF (newpixel) THEN
Accum2 <= Accum2(6 downto 0) & Accum2(20 downto 7);
ELSIF (newbitplane) THEN
Accum2 <= Accum2(19 downto 0) & Accum2(20);
END IF;


IF (input(3) = '1') THEN
Accum3 <= Accum3 + 1;
ELSIF (newblock) THEN
Accum3 <= (others => '0');
ELSIF (newpixel) THEN
Accum3 <= Accum3(6 downto 0) & Accum3(20 downto 7);
ELSIF (newbitplane) THEN
Accum3 <= Accum3(19 downto 0) & Accum3(20);
END IF;


end process;
----

with column select
                avg <=          avg0 when "11",
                                avg1 when "10",
                                avg2 when "01",
                                avg3 when others;   --avg3 is actually the first column.

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE boxaveragepkg IS
COMPONENT boxaverage PORT (
        clock:          IN std_logic;
        input:          IN std_logic_vector(3 downto 0);
        column:         IN std_logic_vector(1 downto 0);
        newpixel:       IN boolean;
        newbitplane:    IN boolean;
        newblock:       IN boolean;
        avg:            OUT std_logic_vector(7 downto 0));
END COMPONENT;
```

```
END boxaveragepkg;
```

B.1.11 Top Level - topwithlights.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/topwithlights.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity topwithlights IS PORT (
        clock:          IN std_logic;  --half clock
        clk:            IN std_logic;   --fast clock
        reset:          IN std_logic;   --normally high, push for low
        index:          IN std_logic_vector(3 downto 0);
        LDbot:          IN std_logic;
        CLR:            IN std_logic;
        avg:            IN std_logic_vector(7 downto 0);
        LDall:          IN std_logic;
        manual:         IN std_logic;  --board switch active low so not it.
        autorange:      IN std_logic;  --board switch active low so not it.
        thresholdlog:   IN std_logic_vector(7 downto 0);
        thresholdlin:   IN std_logic_vector(7 downto 0);
        irisbit:        OUT std_logic;
        modeman:        IN std_logic;           --manual mode input  --board switch active low so not it.
        irisman:        IN std_logic_vector(7 downto 0);
        interin:        IN std_logic_vector(3 downto 0); --in from topbox
        modeled:        OUT std_logic;
        autoled:        BUFFER std_logic;
        autorangeled:   OUT std_logic;
        diffled:        OUT std_logic_vector(7 downto 0);
        frameled:       OUT std_logic_vector(7 downto 0);
        irisled:        OUT std_logic_vector(7 downto 0);
        brightgridled:  OUT std_logic_vector(15 downto 0);      --0 to 15 starting in upper left corner.
        darkgridled:    OUT std_logic_vector(15 downto 0));

attribute pin_avoid of topwithlights:entity is "6 46 76 116";
---------------------------------sclk,smode,SDO,SDI
attribute pin_numbers of topwithlights:entity is
"FRAMELED(0):2 " &
"LDBOT:3 " &
"RESET:4 " &
"AUTORANGE:5 " &
"FRAMELED(1):7 " &
"FRAMELED(2):8 " &
"DARKGRIDLED(0):11 " &
"THRESHOLDLOG(0):12 " &
"INDEX(0):13 " &
"AVG(0):14 " &
"IRISMAN(0):15 " &
"CLOCK:19 " &
"CLK:22 " &
"DARKGRIDLED(1):23 " &
"DARKGRIDLED(2):24 " &
"DARKGRIDLED(3):25 " &
"BRIGHTGRIDLED(0):26 " &
"DARKGRIDLED(4):27 " &
"DARKGRIDLED(5):28 " &
"DARKGRIDLED(6):29 " &
"DARKGRIDLED(7):30 " &
"DARKGRIDLED(8):32 " &
"DARKGRIDLED(9):33 " &
"DARKGRIDLED(10):34 " &
"DARKGRIDLED(11):35 " &
"DARKGRIDLED(12):36 " &
"DARKGRIDLED(13):37 " &
"DARKGRIDLED(14):38 " &
"DARKGRIDLED(15):39 " &
"BRIGHTGRIDLED(1):42 " &
"BRIGHTGRIDLED(2):43 " &
"BRIGHTGRIDLED(3):44 " &
"BRIGHTGRIDLED(4):45 " &
"BRIGHTGRIDLED(5):47 " &
"BRIGHTGRIDLED(6):48 " &
"BRIGHTGRIDLED(7):49 " &
"BRIGHTGRIDLED(8):51 " &
"BRIGHTGRIDLED(9):52 " &
"BRIGHTGRIDLED(10):53 " &
"BRIGHTGRIDLED(11):54 " &
"BRIGHTGRIDLED(12):55 " &
```

```
"BRIGHTGRIDLED(13):56 " &
"BRIGHTGRIDLED(14):57 " &
"BRIGHTGRIDLED(15):58 " &
"THRESHOLDLIN(0):59 " &
"IRISBIT:63 " &
"THRESHOLDLOG(1):64 " &
"AVG(1):65 " &
"AVG(2):66 " &
"IRISMAN(1):67 " &
"THRESHOLDLOG(2):72 " &
"INDEX(1):73 " &
"AVG(3):74 " &
"IRISMAN(2):75 " &
"THRESHOLDLIN(1):79 " &
"IRISLED(0):82 " &
"IRISLED(1):83 " &
"IRISLED(2):84 " &
"IRISLED(3):85 " &
"IRISLED(4):86 " &
"IRISLED(5):87 " &
"IRISLED(6):88 " &
"IRISLED(7):89 " &
"AVG(4):91 " &
"INDEX(2):92 " &
"IRISMAN(3):93 " &
"THRESHOLDLOG(3):97 " &
"THRESHOLDLIN(2):98 " &
"THRESHOLDLIN(3):99 " &
"THRESHOLDLIN(4):102 " &
"IRISMAN(4):104 " &
"AVG(5):105 " &
"INDEX(3):108 " &
"THRESHOLDLOG(4):109 " &
"THRESHOLDLIN(5):110 " &
"DIFFLED(0):112 " &
"DIFFLED(1):113 " &
"DIFFLED(2):115 " &
"DIFFLED(3):118 " &
"DIFFLED(4):119 " &
"DIFFLED(5):122 " &
"FRAMELED(3):123 " &
"DIFFLED(6):124 " &
"DIFFLED(7):125 " &
"FRAMELED(4):126 " &
"FRAMELED(5):127 " &
"FRAMELED(6):128 " &
"FRAMELED(7):129 " &
"THRESHOLDLIN(6):131 " &
"THRESHOLDLOG(5):132 " &
"THRESHOLDLOG(6):133 " &
"AVG(6):134 " &
"IRISMAN(5):135 " &
"MODEMAN:136 " &
"THRESHOLDLIN(7):143 " &
"THRESHOLDLOG(7):144 " &
"LDALL:145 " &
"CLR:146 " &
"IRISMAN(6):147 " &
"INTERIN(0):149 "&
"INTERIN(1):150 "&
"MODELED:152 " &
"AUTOLED:153 " &
"AVG(7):154 " &
"IRISMAN(7):155 " &
"MANUAL:156 " &
"INTERIN(2):157 "&
"INTERIN(3):158 "&
"AUTORANGELED:159 ";

End topwithlights;

USE WORK.std_arith.ALL;

USE WORK.gridlightspkg.all;
USE WORK.topcomponentspkg.all;
USE WORK.lightspkg.all;

ARCHITECTURE behavior OF topwithlights IS

SUBTYPE v2 is std_logic_vector(1 downto 0);
SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);
SUBTYPE v8 is std_logic_vector(7 downto 0);
SUBTYPE v32 is std_logic_vector(31 downto 0);

SIGNAL modechoice: std_logic;
```

```
SIGNAL irischoice: std_logic_vector(7 downto 0);
SIGNAL diff: std_logic_vector(7 downto 0);
SIGNAL frame: std_logic_vector(7 downto 0);
--Attribute synthesis_off of frame: SIGNAL is TRUE;
--Attribute synthesis_off of diff: SIGNAL is TRUE;
--Attribute synthesis_off of irischoice: SIGNAL is TRUE;

SIGNAL greater: boolean;
SIGNAL lesser: boolean;
SIGNAL brightindex: v4;
SIGNAL darkindex: v4;
SIGNAL brightindexp: v4;
SIGNAL darkindexp: v4;
Attribute synthesis_off of greater: SIGNAL is TRUE;
Attribute synthesis_off of lesser: SIGNAL is TRUE;

SIGNAL internaldecision: std_logic;
SIGNAL modedecision: std_logic;

SIGNAL autorangenot: std_logic;
SIGNAL modemannot: std_logic;
SIGNAL resetnot: std_logic;
SIGNAL manualp: std_logic;

BEGIN

topcomponents0: topcomponents PORT MAP(
        clock=>clock,
        clk=>clk,
        reset=>resetnot,
        LDbot=>LDbot,
        CLR=>CLR,
        frameout=>frame,
        diffout=>diff,
        avg=>avg,
        LDall=>LDall,
        manual=> manualp,
        autorange=> autorangenot,
        thresholdlog=>thresholdlog,
        thresholdlin=>thresholdlin,
        irisbit=>irisbit,
        modeman=>modemannot,
        irisman=>irisman,
        modechoice=>modechoice,
        irischoice=>irischoice,
        greater=>greater,
        lesser=>lesser);

lights0: lights PORT MAP(
        clock=>clock,
        frame=>frame,
        diff=>diff,
        LDall=>LDall,
        manual=> manualp,
        autorange=> autorangenot,
        mode=>modechoice,
        iris=>irischoice,
--test
        startframetest=>interin(0),
--
        modeled=>modeled,
        autoled=>autoled,
        autorangeled=>autorangeled,
        diffled=>diffled,
        frameled=>frameled,
        irisled=>irisled);

gridlights: gridlights PORT MAP(
        brightindex=>brightindex,
        darkindex=>darkindex,
        brightgrid=>brightgridled,
        darkgrid=>darkgridled);

resetnot <= not reset;
autorangenot <= not autorange;
modemannot <= not modeman when interin(0) = '1' else interin(2) ;
manualp <= manual when interin(0) = '1' else interin(1) ;

process begin
WAIT UNTIL (clock = '1');

IF LDbot = '1' THEN
IF greater THEN
brightindexp<= index;
END IF;
```

```
IF lesser THEN
darkindexp<= index;
END IF;
END IF;

IF LDall = '1' THEN
brightindex<= brightindexp;
darkindex<=darkindexp;
END IF;

end process;

END behavior;
```

B.1.12 Output Signals - lights.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/lights.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity lights IS PORT (
        clock:          IN std_logic;  --half clock
        frame:          IN std_logic_vector(7 downto 0);
        diff:           IN std_logic_vector(7 downto 0);
        LDall:          IN std_logic;
        manual:         IN std_logic;
        autorange:      IN std_logic;
        mode:           IN std_logic;           --manual mode input
        iris:           IN std_logic_vector(7 downto 0); --test
        startframetest: IN std_logic;
---
        autoled:        BUFFER std_logic;
        modeled:        OUT std_logic;
        autorangeled:   OUT std_logic;
        frameled:       OUT std_logic_vector(7 downto 0);
        irisled:        OUT std_logic_vector(7 downto 0);
        diffled:        OUT std_logic_vector(7 downto 0));
End lights;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF lights IS

SUBTYPE v2 is std_logic_vector(1 downto 0);
SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);
SUBTYPE v5 is std_logic_vector(4 downto 0);
SUBTYPE v8 is std_logic_vector(7 downto 0);

SIGNAL flash: v5;               --counter for dividing Ldall down by 2^4.
constant top: integer:= 4;      --place for msb for flash
SIGNAL framereg: v8;
SIGNAL diffreg: v8;

BEGIN

modeled<= mode;
autoled<= flash(top) when manual = '0' else '0';
autorangeled<= flash(top-1) when autorange = '1' else '0';

--frameled(0)<= '1' when framereg >= "00010000" else '0';
--frameled(1)<= '1' when framereg >= "00110000" else '0';
--frameled(2)<= '1' when framereg >= "01010000" else '0';
--frameled(3)<= '1' when framereg >= "01110000" else '0';
--frameled(4)<= '1' when framereg >= "10010000" else '0';
--frameled(5)<= '1' when framereg >= "10110000" else '0';
--frameled(6)<= '1' when framereg >= "11010000" else '0';
--frameled(7)<= '1' when framereg >= "11110000" else '0';

frameled <= framereg;

--diffled(0)<= '1' when diffreg >= "00010000" else '0';
--diffled(1)<= '1' when diffreg >= "00110000" else '0';
--diffled(2)<= '1' when diffreg >= "01010000" else '0';
--diffled(3)<= '1' when diffreg >= "01110000" else '0';
--diffled(4)<= '1' when diffreg >= "10010000" else '0';
--diffled(5)<= '1' when diffreg >= "10110000" else '0';
--diffled(6)<= '1' when diffreg >= "11010000" else '0';
```

```
--diffled(7)<= '1' when diffreg >= "11110000" else '0';

diffled <= diffreg;

irisled(0)<= '1' when iris >= "00010000" else '0';
irisled(1)<= '1' when iris >= "00110000" else '0';
irisled(2)<= '1' when iris >= "01010000" else '0';
irisled(3)<= '1' when iris >= "01110000" else '0';
irisled(4)<= '1' when iris >= "10010000" else '0';
irisled(5)<= '1' when iris >= "10110000" else '0';
irisled(6)<= '1' when iris >= "11010000" else '0';
irisled(7)<= '1' when iris >= "11110000" else '0';

process begin
WAIT UNTIL (clock = '1');
IF (LDall = '1') THEN
flash <= flash + 1;
framereg <= frame;
diffreg <= diff;
END IF;
end process;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE lightspkg IS
COMPONENT lights PORT (
        clock:          IN std_logic;  --half clock
        frame:          IN std_logic_vector(7 downto 0);
        diff:           IN std_logic_vector(7 downto 0);
        LDall:          IN std_logic;
        manual:         IN std_logic;
        autorange:      IN std_logic;
        mode:           IN std_logic;           --manual mode input
        iris:           IN std_logic_vector(7 downto 0); --test
        startframetest: IN std_logic;
---
        autoled:        BUFFER std_logic;
        modeled:        OUT std_logic;
        autorangeled:   OUT std_logic;
        frameled:       OUT std_logic_vector(7 downto 0);
        irisled:        OUT std_logic_vector(7 downto 0);
        diffled:        OUT std_logic_vector(7 downto 0));
END COMPONENT;
END lightspkg;
```

B.1.13 Next Level - topcomponents.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/topcomponents.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity topcomponents IS PORT (
        clock:          IN std_logic;  --half clock
        clk:            IN std_logic;   --fast clock
        reset:          IN std_logic;
        LDbot:          IN std_logic;
        CLR:            IN std_logic;
        frameout:       OUT std_logic_vector(7 downto 0);
        diffout:        OUT std_logic_vector(7 downto 0);
        avg:            IN std_logic_vector(7 downto 0);
        LDall:          IN std_logic;
        manual:         IN std_logic;
        autorange:      IN std_logic;
        thresholdlog:   IN std_logic_vector(7 downto 0);
        thresholdlin:   IN std_logic_vector(7 downto 0);
        irisbit:        OUT std_logic;
        modeman:        IN std_logic;           --manual mode input
        irisman:        IN std_logic_vector(7 downto 0);
        modechoice:     BUFFER std_logic;
        irischoice:     BUFFER std_logic_vector;
        greater: BUFFER boolean;
        lesser: BUFFER boolean);
End topcomponents;

USE WORK.std_arith.ALL;
```

```
USE WORK.comparepkg.all;
USE WORK.dividerpkg.all;
USE WORK.sendirispkg.all;
USE WORK.irisgenpkg.all;
USE WORK.modegenpkg.all;

ARCHITECTURE behavior OF topcomponents IS

SUBTYPE v2 is std_logic_vector(1 downto 0);
SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);
SUBTYPE v8 is std_logic_vector(7 downto 0);
SUBTYPE v32 is std_logic_vector(31 downto 0);

SIGNAL iris: v8;
SIGNAL irisnext: v8;
SIGNAL dav: std_logic;
SIGNAL irisgo: std_logic;
SIGNAL irisauto: v8;
SIGNAL modenext: std_logic;
SIGNAL difflatched: v8;
SIGNAL toggle: std_logic;
SIGNAL LDalldelayed: std_logic;
SIGNAL framelatched: v8;

SIGNAL modeauto: std_logic;
SIGNAL irisbitprime: std_logic;

SIGNAL frame: v8;
SIGNAL diff: v8;
SIGNAL bright: v8;
Attribute synthesis_off of frame: SIGNAL is TRUE;
Attribute synthesis_off of diff: SIGNAL is TRUE;
Attribute synthesis_off of bright: SIGNAL is TRUE;

BEGIN

compare0: compare PORT MAP(
        clock=>clock,
        CLR=>CLR,
        LDbot=>LDbot,
        frame=>frame,
        diff=>diff,
        bright=>bright,
        avg=>avg,
        greater=>greater,
        lesser=>lesser);

frameout <= frame;
diffout <= diff;

divider0: divider PORT MAP(
        clk=>clock,
        a=>iris,                --a is dividend
        b=>frame,               --b is divisor
        start=>LDall,   --high for one clock to indicate start
        q=>irisnext,    --q is quotient,
        testq=>open,
------
        dav=>dav);      --dav is low until division is done

sendiris0: sendiris PORT MAP(
        clock=>clk,             --fast clock
        dav=>dav,               --high when iris is valid
        irisgo=>irisgo);

irisgen0: irisgen PORT MAP(
        clock=>clk,     --fastclock
        start=>irisgo,
        mode=>modechoice,
        iris=>irischoice,
        irisbit=>irisbitprime);

modgen0: modegen PORT MAP(
        diff=>difflatched,
        frame=>framelatched,
        modecurrent=>modeauto,
        thresholdlog=>thresholdlog,
        thresholdlin=>thresholdlin,
        modenext=>modenext);

modechoice <= modeman when (autorange = '0') ELSE modeauto;
irischoice <= irisman when (manual = '1') ELSE irisauto;
irisauto <= not iris;

irisbit <= irisbitprime or reset;
```

```
process begin
WAIT UNTIL (clock = '1');
IF (LDall = '1') THEN
LDalldelayed <= '1';
difflatched <= diff;
framelatched <= frame;
ELSE LDalldelayed <= '0';
END IF;

IF (LDalldelayed = '1') THEN
        IF toggle = '1' THEN  --can only change modes everyother frame.
                modeauto <= modenext;
                toggle <= '0';
        ELSE toggle <= '1';
        END IF;
END IF;

end process;

process begin
WAIT UNTIL (clock = '1');

IF toggle = '0' THEN  --can only change modes everyother frame.
        IF (dav = '1') THEN
        iris <= irisnext;
        END IF;
END IF;

end process;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE topcomponentspkg IS
COMPONENT topcomponents PORT (
        clock:          IN std_logic;  --half clock
        clk:            IN std_logic;   --fast clock
        reset:          IN std_logic;
        LDbot:          IN std_logic;
        CLR:            IN std_logic;
        frameout:       OUT std_logic_vector(7 downto 0);
        diffout:        OUT std_logic_vector(7 downto 0);
        avg:            IN std_logic_vector(7 downto 0);
        LDall:          IN std_logic;
        manual:         IN std_logic;
        autorange:      IN std_logic;
        thresholdlog:   IN std_logic_vector(7 downto 0);
        thresholdlin:   IN std_logic_vector(7 downto 0);
        irisbit:        OUT std_logic;
        modeman:        IN std_logic;           --manual mode input
        irisman:        IN std_logic_vector(7 downto 0);
        modechoice:     BUFFER std_logic;
        irischoice:     BUFFER std_logic_vector;
        greater: BUFFER boolean;
        lesser: BUFFER boolean);
END COMPONENT;
END topcomponentspkg;
```

B.1.14 LED display for Dynamic Range - gridlights.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/gridlights.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity gridlights IS PORT (
        brightindex:    IN std_logic_vector(3 downto 0);  --first 2 bit are column, second 2 are row.
        darkindex:              IN std_logic_vector(3 downto 0);
        brightgrid:     OUT std_logic_vector(15 downto 0);      --0 to 15 starting in upper left corner.
        darkgrid:               OUT std_logic_vector(15 downto 0));
End gridlights;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF gridlights IS

BEGIN
```

```
process(brightindex,darkindex)

variable brightplace: integer range 0 to 15;
variable darkplace: integer range 0 to 15;

begin

brightplace:= to_integer(brightindex);
darkplace:= to_integer(darkindex);

for i in 15 downto 0 loop
        if i = brightplace then
        brightgrid(i) <= '1';
        else
        brightgrid(i) <= '0';
        end if;
end loop;

for i in 15 downto 0 loop
        if i = darkplace then
        darkgrid(i) <= '1';
        else
        darkgrid(i) <= '0';
        end if;
end loop;

end process;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE gridlightspkg IS
COMPONENT gridlights PORT (
        brightindex:    IN std_logic_vector(3 downto 0);  --first 2 bit are column, second 2 are row.
        darkindex:              IN std_logic_vector(3 downto 0);
        brightgrid:     OUT std_logic_vector(15 downto 0);      --0 to 15 starting in upper left corner.
        darkgrid:               OUT std_logic_vector(15 downto 0));
END COMPONENT;
END gridlightspkg;
```

B.1.15 Iris Generation - sendiris.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/sendiris.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity sendiris IS PORT (
        clock:          IN std_logic;   --fast clock
        dav:            IN std_logic;   --high when iris is valid
        irisgo:         OUT std_logic);
End sendiris;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF sendiris IS

        TYPE states IS (waitfire,waitreset);
        SIGNAL state: states;

BEGIN

PROCESS(clock)
BEGIN

IF clock'EVENT AND clock = '1' THEN

                CASE state IS
                        WHEN waitfire => IF (dav = '1') THEN
                                         irisgo <= '1';
                                         state <= waitreset;
                                         END IF;
                        WHEN waitreset => IF (dav = '0') THEN
                                         state <= waitfire;
                                         END IF;
                                         irisgo <= '0';
                        WHEN OTHERS =>  null;
```

```
                END CASE;
END IF;

END PROCESS;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE sendirispkg IS
COMPONENT sendiris PORT (
        clock:          IN std_logic;   --fast clock
        dav:            IN std_logic;   --high when iris is valid
        irisgo:         OUT std_logic);
END COMPONENT;
END sendirispkg;
```

B.1.16 Comparators - compare.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/compare.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity compare IS PORT (
        clock:          IN std_logic;  --half clock
        CLR:            IN std_logic;
        LDbot:          IN std_logic;
        frame:          OUT std_logic_vector(7 downto 0);
        diff:           OUT std_logic_vector(7 downto 0);
        bright:         BUFFER std_logic_vector(7 downto 0);
        avg:            IN std_logic_vector(7 downto 0);
        greater:        BUFFER boolean;
        lesser:         BUFFER boolean);
End compare;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF compare IS

SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);


SIGNAL frameave: std_logic_vector(12 downto 0);
SIGNAL adder: std_logic_vector(12 downto 0);

SUBTYPE v8 is std_logic_vector(7 downto 0);

SIGNAL  dark: v8;

BEGIN

----
adder <= avg + frameave;
frame <= frameave(11 downto 4) when (frameave(12) = '0') else x"FF";
----

----
PROCESS(avg,clr,bright)
BEGIN
IF (avg > bright) or CLR = '1' THEN
greater <= TRUE;
ELSE
greater <= FALSE;
END IF;
END PROCESS;
----
PROCESS(avg,clr,dark)
BEGIN
IF (avg < dark) or (CLR = '1') THEN
lesser <= TRUE;
ELSE
lesser <= FALSE;
END IF;
END PROCESS;
----

diff <= (bright - dark);
```

```
----

----
Process
BEGIN
wait until clock = '1';
IF (LDbot = '1')THEN

        IF (CLR = '1') THEN
                frameave <= "00000" & avg;
        ELSE
                frameave <= adder;
        END IF;

        IF greater THEN
        bright <= avg;
        END IF;

        IF lesser THEN
        dark <= avg;
        END IF;

END IF;
END Process;
-------

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE comparepkg IS
COMPONENT compare PORT (
        clock:          IN std_logic;  --half clock
        CLR:            IN std_logic;
        LDbot:          IN std_logic;
        frame:          OUT std_logic_vector(7 downto 0);
        diff:           OUT std_logic_vector(7 downto 0);
        bright:         BUFFER std_logic_vector(7 downto 0);
        avg:            IN std_logic_vector(7 downto 0);
        greater:        BUFFER boolean;
        lesser:         BUFFER boolean);
END COMPONENT;
END comparepkg;

--compare0: compare PORT MAP(
--      clock=>clock,
--      CLR=>CLR,
--      LDbot=>LDbot,
--      frame=>frame,
--      diff=>diff,
--      avg=>avg,
--      greater=>greater,
--      lesser=>lesser);
```

B.1.17 Iris Generation - irisgen.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/irisgen.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity irisgen IS PORT (
        clock: IN std_logic;
        start: IN std_logic;        --reset active high
        mode: IN std_logic;
        iris: IN std_logic_vector(7 downto 0);
        irisbit: OUT std_logic);
End irisgen;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF irisgen IS

        SIGNAL count: std_logic_vector(2 downto 0);
        TYPE states IS (idle,high,bits,modestate);
        SIGNAL state: states;

BEGIN
```

```
----
PROCESS (clock,count,start)
        variable order: integer range 0 to 7;
begin

IF clock'EVENT AND clock = '1' THEN
                order:= to_integer(not count);    --send msb first

                CASE state IS
                        WHEN idle =>    IF start = '1' THEN
                                        state <= high;
                                        irisbit <= '1';
                                        ELSE state <= idle;
                                        count <= "000";
                                        irisbit <= '0';
                                        END IF;

                        WHEN high =>    state <= bits;
                                        irisbit <= '0';

                        WHEN bits =>    irisbit <= iris(order);
                                        count <= count + 1;
                                        IF (count = "111") THEN
                                        state <= modestate;
                                        ELSE
                                        state <= bits;
                                        END IF;
                        WHEN modestate =>       irisbit <= mode;
                                        state <= idle;
                        WHEN OTHERS =>  state <= idle;
                END CASE;

END IF;
END PROCESS;
----

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE irisgenpkg IS
COMPONENT irisgen PORT (
        clock: IN std_logic;        --fastclock
        start: IN std_logic;        --reset active high
        mode: IN std_logic;
        iris: IN std_logic_vector(7 downto 0);
        irisbit: OUT std_logic);
END COMPONENT;
END irisgenpkg;
```

B.1.18 Subtraction - sub.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/sub.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;

--USE WORK.numeric_std.all;

Entity sub IS PORT (
        A:              IN std_logic_vector(8 downto 0);
        B:              IN std_logic_vector(7 downto 0);
        co:             OUT std_logic;
        result:         OUT std_logic_vector(7 downto 0));

End sub;

USE WORK.std_arith.ALL;  --found in warp library as mod_gen.vhd
                         --to_integer(a) and to_std_logic_vector(a,size)

ARCHITECTURE behavior OF sub IS

SIGNAL R: std_logic_vector(8 downto 0);

BEGIN

R <= A - ('0' & B);
```

```
result <= R(7 downto 0);
co <= R(8);
--
END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE subpkg IS
COMPONENT sub PORT (
        A:              IN std_logic_vector(8 downto 0);
        B:              IN std_logic_vector(7 downto 0);
        co:             OUT std_logic;
        result:         OUT std_logic_vector(7 downto 0));
END COMPONENT;
END subpkg;
```

B.1.19 Mode Generation - modegen.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/modegen.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;

USE WORK.numeric_std.all;

Entity modegen IS PORT (
        diff:                   IN std_logic_vector(7 downto 0);
        frame:                  IN std_logic_vector(7 downto 0);
        modecurrent:    IN std_logic;
        thresholdlog:   IN std_logic_vector(7 downto 0);
        thresholdlin:   IN std_logic_vector(7 downto 0);
        modenext:               OUT std_logic);
End modegen;

USE WORK.std_arith.ALL;  --found in warp library as mod_gen.vhd
                         --to_integer(a) and to_std_logic_vector(a,size)

ARCHITECTURE behavior OF modegen IS

signal logmode: boolean;

BEGIN

logmode <= modecurrent = '1';

process(diff,thresholdlog,thresholdlin,modecurrent,logmode) begin

IF logmode THEN

        IF (diff < thresholdlog) then modenext <= '0'; END IF;
ELSIF (diff > thresholdlin) then modenext <= '1';
END IF;

--END IF;
end process;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE modegenpkg IS
COMPONENT modegen PORT (
        diff:                   IN std_logic_vector(7 downto 0);
        frame:                  IN std_logic_vector(7 downto 0);
        modecurrent:    IN std_logic;
        thresholdlog:   IN std_logic_vector(7 downto 0);
        thresholdlin:   IN std_logic_vector(7 downto 0);
        modenext:               OUT std_logic);
END COMPONENT;
END modegenpkg;
```

B.1.20 Division - divider.vhd

Location:

/homes/kfife/Imager1/Stereo/ALU/divider.vhd

```
---Divides a by b and multiplies result by 128. q ranges between x00 and xFF. Saturates and cutsoff.
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
use work.subpkg.all;

entity divider is port (
        clk:    in std_logic;
        a:              in std_logic_vector(7 downto 0);        --a is dividend
        b:              in std_logic_vector(7 downto 0);        --b is divisor
        start:  in std_logic;                           --high for one clock to indicate start
        q:              buffer std_logic_vector(7 downto 0);    --q is quotient,
        testq:          buffer std_logic_vector(7 downto 0);    --q is quotient,
        dav:    buffer std_logic);                      --dav is low until division is done
end divider;

architecture behavior of divider is

        signal temp:    std_logic_vector(7 downto 0);   --loads a and shifts out MSB at each cycle
        signal r:       std_logic_vector(7 downto 0);   --remainder
        signal result:  std_logic_vector(7 downto 0);   --result of subtraction
        signal co:      std_logic;                      --carry out
        signal count:   std_logic_vector(3 downto 0);   --determines when 15 cycles have passed
        signal AA:      std_logic_vector(8 downto 0);   --argument for subtraction

begin
process(clk)
begin
        if rising_edge(clk) then

                if start = '1' then
                        dav <= '0';
                        q <= (others => '0');
                        count <= (others => '0');
                        r <= (others => '0');   -- set the remainder to 0 to start
                        temp <= a;              -- set the temp to be the dividend

                elsif dav = '0' then
                        temp <= temp(6 downto 0) & '0';  -- shift out the next bit of dividend
                        if (co = '1') then              --this means that AA was less than b.
                                        q <= q(6 downto 0) & '0';  -- this bit of the quotient is 0
                                r <= r(6 downto 0) & temp(7);    -- set the remainder
                        else
                                q <= q(6 downto 0) & '1';  -- this bit of the quotient is 1
                                        r <= result;  --        set new remainder
                        end if;

                        if q(7) = '1' then              --check to see if q saturates
                                q <= (others => '1');
                                        dav <= '1';
                                elsif (count = "1110") then     --get 8 bits and shift 7 for mult by 128
                                        if "0000100" >= q(6 downto 0) then q <= "00001001"; end if; --this is the lowest we want iris to go
                                        dav <= '1';
                        end if;
                        count <= count + 1;

                end if;
        end if;
end process;

AA <= r(7 downto 0) & temp(7);
-----------subtraction-------------
sub0: sub port map(
        A=>AA,                  --9 bits wide; necessary for certain inputs
        B=>b,                   --8 bits wide;
        co=>co,                 --carry out used to determine whether to load result or shift previous EE
        result=>result);        --8 bits wide;
-------------------------------------

testq <= "01111111";

end behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE dividerpkg IS
```

```
COMPONENT divider PORT (
        clk:    in std_logic;
        a:              in std_logic_vector(7 downto 0);        --a is dividend
        b:              in std_logic_vector(7 downto 0);        --b is divisor
        start:  in std_logic;                                   --high for one clock to indicate start
        q:              buffer std_logic_vector(7 downto 0);    --q is quotient,
        testq:          buffer std_logic_vector(7 downto 0);    --q is quotient,
        dav:    buffer std_logic);                      --dav is low until division is done
END COMPONENT;
END dividerpkg;
k
```

B.1.21 Top Level - monitordigital.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/monitordigital.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity monitor IS PORT (
        clk,clock,reset:IN std_logic;  --reset is not assigned
        clockout:               BUFFER std_logic;  --feed to SRAMS and to clock
        WE,OE:                  OUT std_logic;
        addr:                   BUFFER std_logic_vector(15 downto 0);
        IO:                             INOUT std_logic_vector(7 downto 0);

        WEb,OEb:                OUT std_logic;
        addrb:                  BUFFER std_logic_vector(15 downto 0);
        IOb:                    INOUT std_logic_vector(7 downto 0);
        pixout:                 BUFFER std_logic_vector(7 downto 0);
        dacclk:                 BUFFER std_logic;

        hsync,hblank:   BUFFER std_logic;
        blank:                  OUT std_logic;  --output for DAC blank

        outs,datacall:  IN std_logic;
--interface to pc
        digitalen:              IN      std_logic;
        out0:                   OUT std_logic;  --vertical sync
        BCKCLK:                 OUT std_logic;  --25MHZ
        WEN:                    OUT std_logic;  --active low
        in0:                    IN std_logic;
        in1:                    IN std_logic;
        in2:                    IN std_logic;
        in3:                    IN std_logic;
        pciclk:                 IN std_logic;
--
        interconnect:   OUT std_logic_vector(3 downto 0));  --for other cplds


attribute pin_avoid of monitor:entity is "6 46 76 116";
-----------------------------------sclk,smode,SDO,SDI

attribute pin_numbers of monitor:entity is
"CLOCKOUT:2 " &
"ADDR(0):11 " &
"INTERCONNECT(0):12 " &
"INTERCONNECT(1):13 " &
"ADDRB(0):14 " &
"WEN:16 " &
"INTERCONNECT(2):17 " &
"DATACALL:19 " &
"CLOCK 22 " &
"BCKCLK:23 " &
"IN0:24 " &
"ADDR(1):32 " &
"ADDR(2):33 " &
"ADDRB(1):34 " &
"ADDRB(2):35 " &
"ADDRB(3):36 " &
"ADDR(3):37 " &
"ADDR(4):38 " &
"ADDRB(4):39 " &
"HSYNC:42 " &
"HBLANK:49 " &
"ADDRB(5):51 " &
"ADDR(5):52 " &
"ADDR(6):53 " &
```

```
"ADDRB(6):54 " &
"ADDRB(7):55 " &
"ADDR(7):56 " &
"ADDR(8):57 " &
"ADDRB(8):58 " &
"OUTS:59 " &
"IOb(0):63 " &
"IOb(1):64 " &
"WEb:65 " &
"ADDR(9):66 " &
"ADDR(10):67 " &
"ADDRB(9):68 " &
"ADDRB(10):69 " &
"IOb(2):70 " &
"PIXOUT(0):72 " &
"PIXOUT(1):73 " &
"PIXOUT(2):74 " &
"PIXOUT(3):75 " &
"OE:77 " &
"OEb:78 " &
"DACCLK:79 " &
"IN1:82 " &
"DIGITALEN:83 " &
"IOb(3):91 " &
"WE:92 " &
"IO(0):95 " &
"IOb(4):98 " &
"PCICLK:99 " &
"CLK:102 " &
"IN2:103 " &
"IN3:104 " &
"IO(1):112 " &
"IO(2):113 " &
"IO(3):115 " &
"IO(4):119 " &
"ADDR(11):122 " &
"ADDRB(11):123 " &
"IO(5):124 " &
"IO(6):125 " &
"IO(7):126 " &
"IOb(5):127 " &
"IOb(6):128 " &
"IOb(7):129 " &
"PIXOUT(4):131 " &
"PIXOUT(5):134 " &
"PIXOUT(6):135 " &
"PIXOUT(7):138 " &
"ADDR(12):143 " &
"ADDR(13):144 " &
"ADDR(14):145 " &
"ADDR(15):146 " &
"ADDRB(12):147 " &
"ADDRB(13):148 " &
"ADDRB(14):149 " &
"ADDRB(15):150 " &
"OUTO:152 " &
"INTERCONNECT(3):154 " &
"blank:155 ";

End monitor;

Use work.inopkg.all;
Use work.serial2parallelpkg.all;
Use work.samplepkg.all;
Use work.ntscpkg.all;
Use work.fliplatchpkg.all;
Use work.startlatchpkg.all;

ARCHITECTURE behavioral OF monitor IS

SIGNAL flip: std_logic;
SIGNAL start: std_logic;
SIGNAL datain: std_logic_vector(31 downto 0);
SIGNAL chip: std_logic_vector(31 downto 0);
SIGNAL dav: std_logic;
SIGNAL rdav: std_logic;
SIGNAL pix: std_logic;
SIGNAL startf: std_logic;
SIGNAL startr: std_logic;
SIGNAL done: std_logic;
SIGNAL startreset: std_logic;
SIGNAL one: std_logic := '1';
SIGNAL low: std_logic := '0';

SIGNAL samplepixel: boolean;
SIGNAL vertsync: std_logic;
```

```
BEGIN

serial2parallel0: serial2parallel PORT MAP(
        clk=>clk,
        start=>start,
        data=>datain,
        outs=>outs,
        dav=>dav,
        rdav=>rdav,
        datacall=>datacall);

sample0: sample PORT MAP(
        clock=>clock,
        dav=>dav,
        rdav=>rdav,
        samplenow=>pix,
        datain=>datain,
        dataout=>chip);

fliplatch0: fliplatch PORT MAP(
        clock=>clock,
        startreset=>startreset,
        flip=>flip);

startlatch0: startlatch PORT MAP(
        clock=>clock,
        start=>start,
        done=>done,
        startreset=>startreset);

ino0: ino PORT MAP(
        clock=>clock,
        done=>done,
        WE=>WE,
        OE=>OE,
        addr=>addr,
        chip=>chip,
        IO=>IO,
        pix=>pix,
        start=>startreset,
        startf=>startf,
        startr=>startr,
        WE1=>WEb,
        OE1=>OEb,
        addr1=>addrb,
        IO1=>IOb,
        pixout=>pixout,
        dacclk=>dacclk,
        --
        samplepixel=>samplepixel,       --boolean
        vertsync=>vertsync,             --std_logic
        --
        flip=>flip);

ntsc0: ntsc PORT MAP(
        clock=>clock,
        startreset=>startreset,
        hsync=>hsync,
        hblank=>hblank,
        startr=>startr,
        startf=>startf);

--the pixout is wired to the header
--pciclk and digitalen are not driven;
out0<=vertsync;
BCKCLK<=dacclk;
WEN <= '0' when samplepixel and (flip = '1' or in3 = '1') else '1';
--
interconnect <= in3 & in2 & in1 & in0;

process begin
WAIT UNTIL clk = '1';
clockout <= not clockout;   --divides the clk by 2.
end process;

blank <= not hblank;  --for the dac blanking

END behavioral;
```

B.1.22 Serial-to-Parallel Converter - serial2parallel.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/serial2parallel.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity serial2parallel IS PORT (
        clk:            IN std_logic;           --clk on board receiving end
        start:          BUFFER std_logic;       --start signal for the start of frame
        data:           BUFFER std_logic_vector(31 downto 0);
        outs:           IN std_logic;           --bit input from imager
        dav:            OUT std_logic;          --indication of when data is ready
        rdav:           IN std_logic;           --resets dav asynchronously
        datacall:       IN std_logic);          --clock for outs start signal embedded
End serial2parallel;

USE WORK.std_arith.ALL;

ARCHITECTURE datapath OF serial2parallel IS

SIGNAL count: std_logic_vector(4 downto 0);     --clocked with datacall
SIGNAL ckcount: std_logic_vector(2 downto 0);   --counter for check for deadzone in datacall
SIGNAL check: std_logic_vector(4 downto 0);     --used to sample count value
SIGNAL previouscheck: std_logic_vector(4 downto 0);     --used to sample check value

SIGNAL breakcondition: boolean;
SIGNAL idlecondition: boolean;
SIGNAL normalcondition: boolean;

SIGNAL resetcount: std_logic;
TYPE states IS (normal,break,waitsample,idle);
SIGNAL state : states;

BEGIN

--------serial to parallel conversion-----------
PROCESS(datacall,rdav,resetcount,start)
begin

IF rdav = '1' THEN
dav <= '0';

ELSIF (resetcount = '1' or start = '1') THEN
count <= (others => '0');

ELSIF datacall'EVENT AND datacall = '1' THEN

        data <= data(30 downto 0) & outs;
        count <= count + 1;

        IF (count = "11111") THEN
        dav <= '1';
        END IF;
END IF;
END PROCESS;
-------------------------------------------------

---decoding the start frame signal-------------
--The process samples the count value with check. Check is sampled by previouscheck.
--By comparing the two values, the process determines when there is a break in the
--datacall clock. It then looks for the start signal by probing check and previouscheck
--during a 4 period clock sequence. If found, the start signal is fired. Otherwise the
--process resumes normal behavior.


breakcondition <= (ckcount = "011") and (previouscheck = check);
idlecondition <= (ckcount = "111") and (previouscheck = check);
normalcondition <= (check(0) = '1');

process(clk)
begin

IF rising_edge(clk) THEN
        check <= count;
        previouscheck <= check;
        IF (previouscheck = check) THEN
        ckcount <= ckcount + 1;
        ELSE
        ckcount <= (others => '0');
        END IF;
```

```
        CASE state is
                        when normal =>  start <= '0';
                                                resetcount <= '0';
                                        IF breakcondition THEN
                                                state <= break;
                                        END IF;
                        when break =>
                                        IF idlecondition THEN
                                                resetcount <= '1';
                                                state <= waitsample;
                                        ELSIF (previouscheck /= check) THEN
                                                start <= '1';
                                                state <= waitsample;
                                        END IF;
                        when waitsample =>      resetcount <= '0';
                                                        start <= '0';
                                                        state <= idle;
                        when idle =>
                                        IF normalcondition THEN
                                                state <= normal;
                                        END IF;
                        when others => state <= normal;
        END CASE;
END IF;
end process;
-------------------------------------

END datapath;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE serial2parallelpkg IS
COMPONENT serial2parallel PORT (
        clk:            IN std_logic;           --clk on board receiving end
        start:          BUFFER std_logic;       --start signal for the start of frame
        data:           BUFFER std_logic_vector(31 downto 0);
        outs:           IN std_logic;           --bit input from imager
        dav:            OUT std_logic;          --indication of when data is ready
        rdav:           IN std_logic;           --resets dav asynchronously
        datacall:       IN std_logic);          --clock for outs start signal embedded
END COMPONENT;
END serial2parallelpkg;
```

B.1.23 Format Converter - inout.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/inout.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity ino IS PORT (
        clock:  IN std_logic;
        done:   OUT std_logic;
        WE,OE:  OUT std_logic;
        addr:   BUFFER std_logic_vector(15 downto 0);
        chip:   IN std_logic_vector(31 downto 0);
        IO:     INOUT std_logic_vector(7 downto 0);
        pix,start: IN std_logic; --start indicates begining of frame
                                 --pix indicates when a set of 32 bits from the imager are ready
        startf,startr: IN std_logic;    --startf causes pixels to be output at begining of frame
        WE1,OE1: OUT std_logic;         --starfr cause pixels to be output at begining of row
        addr1:  BUFFER std_logic_vector(15 downto 0);
        IO1:    INOUT std_logic_vector(7 downto 0);
        pixout: BUFFER std_logic_vector(7 downto 0);

        dacclk: BUFFER std_logic;
        samplepixel: OUT boolean;
        vertsync: OUT std_logic;

        flip:   IN std_logic);   --flip switches SRAM parameters. ie- addr,WE,OE,IO

End ino;


USE WORK.std_arith.ALL;

ARCHITECTURE fsmino OF ino IS
```

```
        TYPE states IS (idle,doread,dowrite);
        SIGNAL state : states;

        SUBTYPE v17 is std_logic_vector(16 downto 0);
        SUBTYPE v8 is std_logic_vector(7 downto 0);
        SUBTYPE v2 is std_logic_vector(1 downto 0);
        SUBTYPE v3 is std_logic_vector(2 downto 0);

        SIGNAL row_prime_part_n_inc_pac: v17;

        alias pac: v3 is                row_prime_part_n_inc_pac(2 downto 0);
        alias inc: v2 is                row_prime_part_n_inc_pac(4 downto 3);
        alias n: std_logic is           row_prime_part_n_inc_pac(5);
        alias part: v2 is               row_prime_part_n_inc_pac(7 downto 6);
        alias prime: std_logic is       row_prime_part_n_inc_pac(8);
        alias row: v8 is                row_prime_part_n_inc_pac(16 downto 9);

        SIGNAL buff: std_logic_vector(3 downto 0);
---------------------------------
        TYPE states1 IS (rowwait, dorow);
        SIGNAL state1 : states1;

        SIGNAL count18: std_logic_vector(17 downto 0);

        alias row1: std_logic_vector(7 downto 0) is count18(17 downto 10);
        alias pac1: std_logic_vector(2 downto 0) is count18(9 downto 7);
        alias x: std_logic is count18(6);
        alias inc1: std_logic_vector(1 downto 0) is count18(5 downto 4);
        alias prime1: std_logic is count18(3);
        alias y: std_logic is count18(2);
        alias part1: std_logic_vector(1 downto 0)is count18(1 downto 0);

        SIGNAL startflag: std_logic;
        SIGNAL top: integer range 4 to 7;
        SIGNAL pixel: std_logic_vector(7 downto 0);

        SIGNAL lastsample: boolean;
        SIGNAL firstsample: std_logic;
        SIGNAL firstprime: boolean;
        SIGNAL dontsample: boolean;
        SIGNAL samplepixelp: boolean;

-------statemachine for input formatting-----------------
BEGIN
PROCESS (clock,pix,inc,n,part,prime,row,start)
BEGIN

IF (start = '1') THEN
        state <= idle;
        row_prime_part_n_inc_pac <= (others => '0');

        ELSIF clock'EVENT AND clock = '1' THEN
                CASE state IS
                        WHEN idle =>    done <= '0';
                                        IF (pix = '1') THEN
                                        state <= doread;
                                        END IF;
                        WHEN doread => state <= dowrite;
                        WHEN dowrite =>
                                        IF (pac = "111") THEN
                                        state <= idle;
                                        done <= '1';
                                        ELSE
                                        state <= doread;
                                        END IF;
                                        row_prime_part_n_inc_pac <= row_prime_part_n_inc_pac + 1;
                        WHEN OTHERS => state <= idle;
                END CASE;
        END IF;
END PROCESS;
-----------------------------------------------------------------

-------controlling WE and OE--------------------
PROCESS (state,flip,state1,count18)
BEGIN

if flip = '0' then
        if (state = dowrite) then
                WE <= '0';
        else WE <= '1';
        end if;

        if (state = doread) then
                OE <= '0';
        else OE <= '1';
        end if;
```

```
        if (state1 = dorow) then        ---from read process
        OE1 <= '0';
        else
        OE1 <= '1';
        end if;
        WE1 <= '1';
else
        if (state = dowrite) then
                WE1 <= '0';
        else WE1 <= '1';
        end if;

        if (state = doread) then
                OE1 <= '0';
        else OE1 <= '1';
        end if;

        if state1 = dorow then          ---from read process
        OE <= '0';
        else
        OE <= '1';
        end if;
        WE <= '1';
end if;
END PROCESS;
-------------------------------------------------

-------------controlling IO ports----------------
PROCESS (state,clock,buff,IO,IO1,flip)
BEGIN
IF clock'EVENT AND clock = '1' THEN
        if flip = '0' then
                IF (state = doread) THEN  -- read out of SRAM and put the value in buff
                buff <= IO(3 downto 0);
                END IF;
        else
                IF (state = doread) THEN  -- read out of SRAM1 and put the value in buff
                buff <= IO1(3 downto 0);
                END IF;
        end if;
END IF;
END PROCESS;

PROCESS (state,buff,chip,flip,clock,pac)
variable index0: integer range 0 to 31;
variable index1: integer range 0 to 31;
variable index2: integer range 0 to 31;
variable index3: integer range 0 to 31;
BEGIN
index0 := to_integer(pac) * 4;
index1 := to_integer(pac) * 4 + 1;
index2 := to_integer(pac) * 4 + 2;
index3 := to_integer(pac) * 4 + 3;

        if flip = '0' then
        IO1 <= "ZZZZZZZZ";                      --allows IO1 to be used as an input during this time
                IF (state = dowrite) THEN       --read out of SRAM and put the value in buff
                IO <= buff & chip(index0) & chip(index1) & chip(index2) & chip(index3);
                ELSE                            --in buff and write to SRAM
                IO <= "ZZZZZZZZ";
                END IF;
        else
        IO <= "ZZZZZZZZ";
                IF (state = dowrite) THEN       --read out of SRAM and put the value in buff
                IO1 <= buff & chip(index0) & chip(index1) & chip(index2) & chip(index3);
                ELSE                            --in buff and write to SRAM
                IO1 <= "ZZZZZZZZ";
                END IF;
        end if;

END PROCESS;
-----------------------------------------

---------------RASTER------OUTPUT----------------
PROCESS (clock,state1,startf,IO1,pixel,startr,flip)

        VARIABLE bot: integer range 0 to 3:= top - 4;

BEGIN

        IF      (startf = '1') THEN
                firstprime <= true;
                state1 <= rowwait;
                count18 <= (others => '0');
```

```
        ELSIF clock'EVENT AND clock = '1' THEN

                CASE state1 IS

                        WHEN dorow =>
                                if flip = '0' then
                                        CASE part1 IS    -- read out of SRAM
                                                WHEN "00" => pixel(7 downto 6) <= IO1(top) & IO1(bot);
                                                WHEN "01" => pixel(5 downto 4) <= IO1(top) & IO1(bot);
                                                WHEN "10" => pixel(3 downto 2) <= IO1(top) & IO1(bot);
                                                WHEN "11" => pixel(1 downto 0) <= IO1(top) & IO1(bot);
                                                WHEN OTHERS => pixel <= pixel;
                                        END CASE;
                                else
                                        CASE part1 IS    -- read out of SRAM
                                                WHEN "00" => pixel(7 downto 6) <= IO(top) & IO(bot);
                                                WHEN "01" => pixel(5 downto 4) <= IO(top) & IO(bot);
                                                WHEN "10" => pixel(3 downto 2) <= IO(top) & IO(bot);
                                                WHEN "11" => pixel(1 downto 0) <= IO(top) & IO(bot);
                                                WHEN OTHERS => pixel <= pixel;
                                        END CASE;

                                end if;

                                        CASE part1 IS    -- load dac
                                                WHEN "00" => pixout <= pixel;
                                                                        vertsync <= firstsample;
                                                                    --for digital out header--
                                                                    IF firstprime THEN
                                                                    firstsample <= '1';
                                                                    firstprime <= false;
                                                                    ELSE firstsample <= '0';
                                                                    END IF;

                                                                    IF dontsample THEN
                                                                dontsample <= false;
                                                                    ELSE
                                                                    samplepixelp <= true;
                                                                    END IF;
                                                                    ------
                                                WHEN OTHERS => pixout <= pixout;
                                                                    samplepixelp <= false;
                                        END CASE;

                                        IF (count18(9 downto 0) = "1111111111") THEN
                                        state1 <= rowwait;
                                        ELSE
                                        state1 <= dorow;
                                        END IF;

                                        count18 <= count18 + 1;

                        WHEN rowwait => count18 <= count18;
                                        pixout <= pixel;
                                        IF lastsample THEN
                                                samplepixelp <= true;
                                                lastsample <= false;
                                        ELSE samplepixelp <= false;
                                        END IF;

                                        IF startr = '1' THEN
                                                state1 <= dorow;
                                                lastsample <= true;
                                                dontsample <= true;
                                        END IF;

                        WHEN OTHERS => state1 <= rowwait;
                END CASE;


        END IF;
END PROCESS;

top <= 7 - to_integer(x & y);

dacclk <= not clock;   -- or just clock;

----------------output addressing-----------------------
process(flip,row,prime,part,pac,inc,row1,prime1,part1,pac1,inc1)
begin
if flip = '0' then
addr <= row & prime & part & pac & inc;
addr1 <=  row1 & prime1 & part1 & pac1 & inc1;
else
addr1 <= row & prime & part & pac & inc;
addr <=  row1 & prime1 & part1 & pac1 & inc1;
```

```
end if;
end process;

-------------------------------------------------------

samplepixel <= samplepixelp;

END fsmino;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE inopkg IS
COMPONENT ino PORT (
        clock: IN std_logic;
        done: OUT std_logic;
        WE,OE: OUT std_logic;
        addr: BUFFER std_logic_vector(15 downto 0);
        chip: IN std_logic_vector(31 downto 0);
        IO: INOUT std_logic_vector(7 downto 0);
        pix,start: IN std_logic; --start indicates begining of frame
                                 --pix indicates when a set of 32 bits from the imager are ready
        startf,startr: IN std_logic;   --startf causes pixels to be output at begining of frame
        WE1,OE1: OUT std_logic;        --starfr cause pixels to be output at begining of row
        addr1: BUFFER std_logic_vector(15 downto 0);
        IO1: INOUT std_logic_vector(7 downto 0);
        pixout: BUFFER std_logic_vector(7 downto 0);

        dacclk: BUFFER std_logic;
        samplepixel: OUT boolean;
        vertsync: OUT std_logic;

        flip: IN std_logic);   --flip switches SRAM parameters. ie- addr,WE,OE,IO
END COMPONENT;
END inopkg;
```

B.1.24 Synchronous Sample - sample.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/sample.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity sample IS PORT (
        clock:          IN std_logic;
        dav:            IN std_logic;
        rdav:           BUFFER std_logic;
        samplenow:      OUT std_logic;
        datain:         IN std_logic_vector(31 downto 0);
        dataout:        OUT std_logic_vector(31 downto 0));
End sample;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF sample IS

signal data: std_logic_vector(31 downto 0);

BEGIN

process begin
WAIT UNTIL (clock = '1');
IF dav = '1' THEN
rdav <= '1';
ELSE
rdav <= '0';
END IF;
end process;

PROCESS(clock,dav,datain)

BEGIN

IF clock'EVENT AND clock = '1' THEN

                IF rdav = '1' THEN
                data <= datain;
                samplenow <= '1';
                ELSE
                samplenow <= '0';
```

```
                END IF;
END IF;

END PROCESS;

dataout <=
data(0) & data(4) & data(8) & data(12)&
data(16) & data(20) & data(24) & data(28)&
data(1) & data(5) & data(9) & data(13)&
data(17) & data(21) & data(25) & data(29)&
data(2) & data(6) & data(10) & data(14)&
data(18) & data(22) & data(26) & data(30)&
data(3) & data(7) & data(11) & data(15)&
data(19) & data(23) & data(27) & data(31);

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE samplepkg IS
COMPONENT sample PORT (
        clock:          IN std_logic;
        dav:            IN std_logic;
        rdav:           BUFFER std_logic;
        samplenow:      OUT std_logic;
        datain:         IN std_logic_vector(31 downto 0);
        dataout:        OUT std_logic_vector(31 downto 0));
END COMPONENT;
END samplepkg;
```

B.1.25 NTSC Encoder - ntsc.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/ntsc.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity ntsc IS PORT (
        clock:          IN std_logic;   --reset active low
        startreset:     IN std_logic;
        hsync,hblank:   BUFFER std_logic;
        startr,startf:  OUT std_logic);
End ntsc;

USE WORK.std_arith.ALL;

ARCHITECTURE fsmntsc OF ntsc IS
        TYPE states IS (visible,EQ1,ser,EQ2,blank);
        SIGNAL state : states;

        SIGNAL count: integer range 0 to 1599;
        SIGNAL line: integer range 0 to 255;
----parameters
        CONSTANT fpw: integer:= 192;            --front porch
        CONSTANT bpw: integer:= 256;    --back porch
        CONSTANT syncw: integer:= 128;          --sync width
        CONSTANT serw: integer:= 112;   --seration
        CONSTANT ew: integer:= 64;              --equalization
        CONSTANT visible_lines: integer:= 242;
        CONSTANT rowclk: integer := 1600;
----------
        constant serl: integer:= rowclk/2 - serw;
        constant halfrow: integer:= rowclk/2;

        constant ew2:integer:= halfrow + ew;
        constant serl2: integer:= rowclk - serw;

        constant bp_blank: integer:= syncw + bpw;
        constant fp_blank: integer:= rowclk - fpw;
        constant start_read: integer:= bp_blank - 6;

        SIGNAL visible_region: boolean;
        SIGNAL eq_region: boolean;
        SIGNAL ser_region: boolean;
        SIGNAL blank_region: boolean;

BEGIN

visible_region <= state = visible;
```

```
eq_region <= state = EQ1 or state = EQ2;
ser_region <= state = ser;
blank_region <= state = blank;

PROCESS (clock,count,startreset,state)


BEGIN

        IF clock'EVENT AND clock = '1' THEN

                --Eq and Ser pulses

                IF ((line = 255) AND (count = 1599)) or (startreset = '1') THEN
                count <= 0;
                line <= 0;
                state <= visible;

                ELSIF (count = 1599) THEN
                count <= 0;
                line <= line + 1;

                ELSE count <= count + 1;
                END IF;


                CASE state IS
                        WHEN visible => IF line = (visible_lines) THEN
                                        state <= EQ1;
                                        END IF;
                        WHEN EQ1 =>     IF line = (visible_lines + 3) THEN
                                        state <= ser;
                                        END IF;
                        WHEN ser =>     IF line = (visible_lines + 6) THEN
                                        state <= EQ2;
                                        END IF;
                        WHEN EQ2 =>     IF line = (visible_lines + 9) THEN
                                        state <= blank;
                                        END IF;
                        WHEN blank =>   IF line = 0 THEN
                                        state <= visible;
                                        END IF;
                        WHEN OTHERS => state <= state;
                END CASE;

                CASE count IS
                        WHEN 0 =>       hsync <= '0';   --start hsync pulse

                        WHEN syncw =>   IF visible_region or blank_region THEN
                                        hsync <= '1';  --end hsync pulse
                                        END IF;
                        WHEN ew =>      IF eq_region THEN --eq width
                                        hsync <= '1';   --eq pulse width
                                        END IF;
                        WHEN ew2 =>     IF eq_region THEN --eq width
                                        hsync <= '1';   --eq pulse width
                                        END IF;
                        WHEN halfrow => IF eq_region or ser_region THEN
                                        hsync <= '0';
                                        END IF;
                        WHEN serl =>    IF ser_region THEN  --ser width
                                        hsync <= '1';
                                        END IF;
                        WHEN serl2 =>   IF ser_region THEN  --ser width
                                        hsync <= '1';
                                        END IF;
                        WHEN OTHERS => hsync <= hsync;
                END CASE;

                CASE count IS

                        WHEN bp_blank => IF visible_region THEN
                                         hblank <= '1';  --active video after back porch
                                         ELSE hblank <= '0';
                                         END IF;
                        WHEN fp_blank =>hblank <= '0';  --front porch start

                        WHEN OTHERS => hblank <= hblank;
                END CASE;

                IF (count = start_read) THEN
                startr <= '1'; ELSE startr <= '0';
                END IF;

                IF (count = start_read - 4) and (line = 0) THEN
```

```
            startf<= '1'; ELSE startf <= '0';
            END IF;

      END IF;
END PROCESS;

END fsmntsc;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE ntscpkg IS
COMPONENT ntsc PORT (
      clock:          IN std_logic;   --reset active low
      startreset:     IN std_logic;
      hsync,hblank:   BUFFER std_logic;
      startr,startf:  OUT std_logic);
END COMPONENT;
END ntscpkg;
```

B.1.26 Generate *flip* Signal - flip.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/flip.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity fliplatch IS PORT (
      clock:          IN std_logic;
      startreset: IN std_logic;
      flip:           BUFFER std_logic);
End fliplatch;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF fliplatch IS

BEGIN

PROCESS(clock)

BEGIN

IF clock'EVENT AND clock = '1' THEN
      IF startreset = '1' THEN
            flip <= not flip;
      END IF;
END IF;

END PROCESS;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE fliplatchpkg IS
COMPONENT fliplatch PORT (
      clock:          IN std_logic;
      startreset: IN std_logic;
      flip:           BUFFER std_logic);
END COMPONENT;
END fliplatchpkg;
```

B.1.27 Start of Frame - startlatch.vhd

Location:

/homes/kfife/Imager1/Stereo/DATAPATH/startlatch.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity startlatch IS PORT (
        clock:          IN std_logic;
        start:          IN std_logic;
        done:           IN std_logic;
        startreset:     BUFFER std_logic);
End startlatch;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF startlatch IS

        TYPE states IS (ready,go);
        SIGNAL state: states;
        SIGNAL startQ: std_logic; --internal signal for rs latch.
BEGIN

--grab asynchronous start--

--Q <= not(reset or not(set or Q));  --rs latch

startQ <= not(startreset or not(start or startQ));

PROCESS(clock)

BEGIN

IF clock'EVENT AND clock = '1' THEN

                CASE state IS
                        WHEN ready =>   IF (startQ = '1') THEN
                                        state <= go;
                                        END IF;
                                        startreset <= '0';
                        WHEN go =>      IF (done = '1') THEN
                                        startreset <= '1';
                                        state <= ready;
                                        END IF;
                        WHEN OTHERS =>  state <= ready;
                END CASE;

END IF;

END PROCESS;

END behavior;

Library ieee;
Use ieee.std_logic_1164.all;
PACKAGE startlatchpkg IS
COMPONENT startlatch PORT (
        clock:          IN std_logic;
        start:          IN std_logic;
        done:           IN std_logic;
        startreset:     BUFFER std_logic);
END COMPONENT;
END startlatchpkg;
```

B.1.28 Test Interface to GuPPI Card - topguppi.vhd

Location:

/homes/kfife/Imager1/Stereo/GUPPI/topguppi.vhd

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity guppi IS PORT (
        clock:          IN std_logic;  --registers

        ins:            OUT std_logic_vector(31 downto 0);  --top
        inff:           IN std_logic;
        bckclk:         OUT std_logic;
        inaf:           IN std_logic;
        inhf:           IN std_logic;
        inae:           IN std_logic;
        inef:           IN std_logic;
        wen:            OUT std_logic;

        bcken:          IN std_logic;                                   --side
```

```
        pciclk:         IN std_logic;                                   --registers
        bckrs:          IN std_logic;
        bckid:          OUT std_logic_vector(3 downto 0);
        bckctrl:        IN std_logic_vector(3 downto 0);
        bcksts:         OUT std_logic_vector(3 downto 0);

        pixout:         IN std_logic_vector(7 downto 0);  --digitalhead
        out0:           IN std_logic;
        pciclkred:      OUT std_logic;
        inred:          OUT std_logic_vector(3 downto 0);
        wenred:         IN std_logic;
        bckclkred:      IN std_logic;

        byteA:          IN std_logic_vector(7 downto 0);  --on board
        byteB:          IN std_logic_vector(7 downto 0);
        byteC:          IN std_logic_vector(7 downto 0);

        led:            OUT std_logic_vector(7 downto 0)); --on board leds

attribute pin_avoid of guppi:entity is "6 46 76 116";
---------------------------------sclk,smode,SDO,SDI

attribute pin_numbers of guppi:entity is
"BCKID(0):2 " &
"INRED(0):3 " &
"INRED(1):4 " &
"INRED(2):5 " &
"INRED(3):7 " &
"INS(0):8 " &
"INS(1):9 " &
"BCKID(1):11 " &
"BCKSTS(0):12 " &
"BCKSTS(1):13 " &
"BCKCLK:14 " &
"PCICLKRED:15 " &
"WEN:16 " &
"BCKSTS(2):17 " &
"CLOCK:19 " &
"PCICLK:22 " &
"INS(2):23 " &
"BCKCLKRED:27 " &
"BCKCTRL(0):28 " &
"BYTEB(0):29 " &
"INFF:30 " &
"INS(3):32 " &
"BCKCTRL(1):35 " &
"BYTEA(2):36 " &
"BYTEA(3):37 " &
"BYTEB(1):38 " &
"INAE:39 " &
"INS(4):42 " &
"BCKCTRL(2):45 " &
"BYTEC(2):47 " &
"BYTEA(4):48 " &
"PIXOUT(7):49 " &
"INS(5):51 " &
"BCKCTRL(3):54 " &
"BYTEC(3):55 " &
"BYTEA(5):56 " &
"PIXOUT(4):57 " &
"PIXOUT(0):58 " &
"PIXOUT(1):59 " &
"INS(6):63 " &
"BCKEN:66 " &
"BYTEC(4):67 " &
"BYTEA(6):68 " &
"PIXOUT(5):69 " &
"PIXOUT(6):70 " &
"INS(7):72 " &
"INEF:75 " &
"BYTEC(5):77 " &
"BYTEA(7):78 " &
"BYTEB(2):79 " &
"INS(8):82 " &
"OUT0:85 " &
"BYTEC(6):86 " &
"BYTEB(3):87 " &
"BYTEB(4):88 " &
"INHF:89 " &
"INS(9):91 " &
"WENRED:94 " &
"BYTEC(7):95 " &
"BYTEB(5):96 " &
"BYTEB(6):97 " &
"INAF:98 " &
"PIXOUT(2):99 " &
```

```
"PIXOUT(3):102 " &
"INS(10):103 " &
"LED(0):107 " &
"BYTEC(0):108 " &
"BYTEA(0):109 " &
"BYTEB(7):110 " &
"INS(11):112 " &
"LED(1):113 " &
"BYTEC(1):118 " &
"BYTEA(1):119 " &
"INS(12):122 " &
"BCKSTS(3):124 " &
"LED(2):125 " &
"INS(13):126 " &
"BCKRS:127 " &
"LED(3):128 " &
"INS(14):131 " &
"LED(4):132 " &
"LED(5):133 " &
"LED(6):134 " &
"INS(15):135 " &
"LED(7):136 " &
"INS(16):137 " &
"INS(17):138 " &
"BCKID(2):143 " &
"INS(18):144 " &
"INS(19):145 " &
"INS(20):146 " &
"INS(21):147 " &
"INS(22):148 " &
"INS(23):149 " &
"INS(24):150 " &
"BCKID(3):152 " &
"INS(25):153 " &
"INS(26):154 " &
"INS(27):155 " &
"INS(28):156 " &
"INS(29):157 " &
"INS(30):158 " &
"INS(31):159 ";
End guppi;

USE WORK.std_arith.ALL;

ARCHITECTURE behavior OF guppi IS

SUBTYPE v1 is std_logic;
SUBTYPE v2 is std_logic_vector(1 downto 0);
SUBTYPE v4 is std_logic_vector(3 downto 0);
SUBTYPE v3 is std_logic_vector(2 downto 0);
SUBTYPE v8 is std_logic_vector(7 downto 0);
SUBTYPE v32 is std_logic_vector(31 downto 0);

SIGNAL truebyteA: boolean;
SIGNAL truebyteB: boolean;
SIGNAL truebyteC: boolean;
--Attribute synthesis_off of halfbyte: SIGNAL is TRUE;
--Attribute synthesis_off of lesser: SIGNAL is TRUE;

BEGIN

truebyteA <= byteA = x"FF";
truebyteB <= byteB = x"FF";
truebyteC <= byteC = x"FF";

led <= bckctrl & "0000";

process begin
WAIT UNTIL (pciclk = '1');
bckid <= "0001";
end process;

bcksts <= "ZZZZ";

ins <= x"FF" & pixout & pixout & pixout when (out0 = '1') else x"00" & pixout & pixout & pixout;
bckclk <= bckclkred;
wen <= wenred;
inred <= bckctrl;

--unimportant
pciclkred <= pciclk;
--

END behavior;
```

B.2 Schematics

The board schematics, library files, and PC board layout for all parts of the Automatic Brightness Adapation demonstration are located in `/homes/kfife/Accel` in ACCEL binary format.

Schematics for the Digital Imager are shown in Figures B.1 - Figures B.2. The corresponding files are located in:

```
/homes/kfife/Accel/Imager/
        combined.sch
        combined.pcb
        imager1.pcb
        imager2.lib
```

Schematics for the Automatic Brightness Adaption Board are shown in Figures B.3 - Figures B.5. The corresponding files are located in:

```
/homes/kfife/Accel/Autobright/
        box.sch
        box.pcb
        autobright.pcb
        imager2.lib
        autobright.lib
```

The schematic for the Tadpole Daughter Card is shown in Figure B.6.The corresponding files are located in:

```
/homes/kfife/Accel/Guppi/
        guppi.sch
        guppi.pcb
        Rguppi.pcb
        guppi.lib
        imager2.lib
        autobright.lib
```

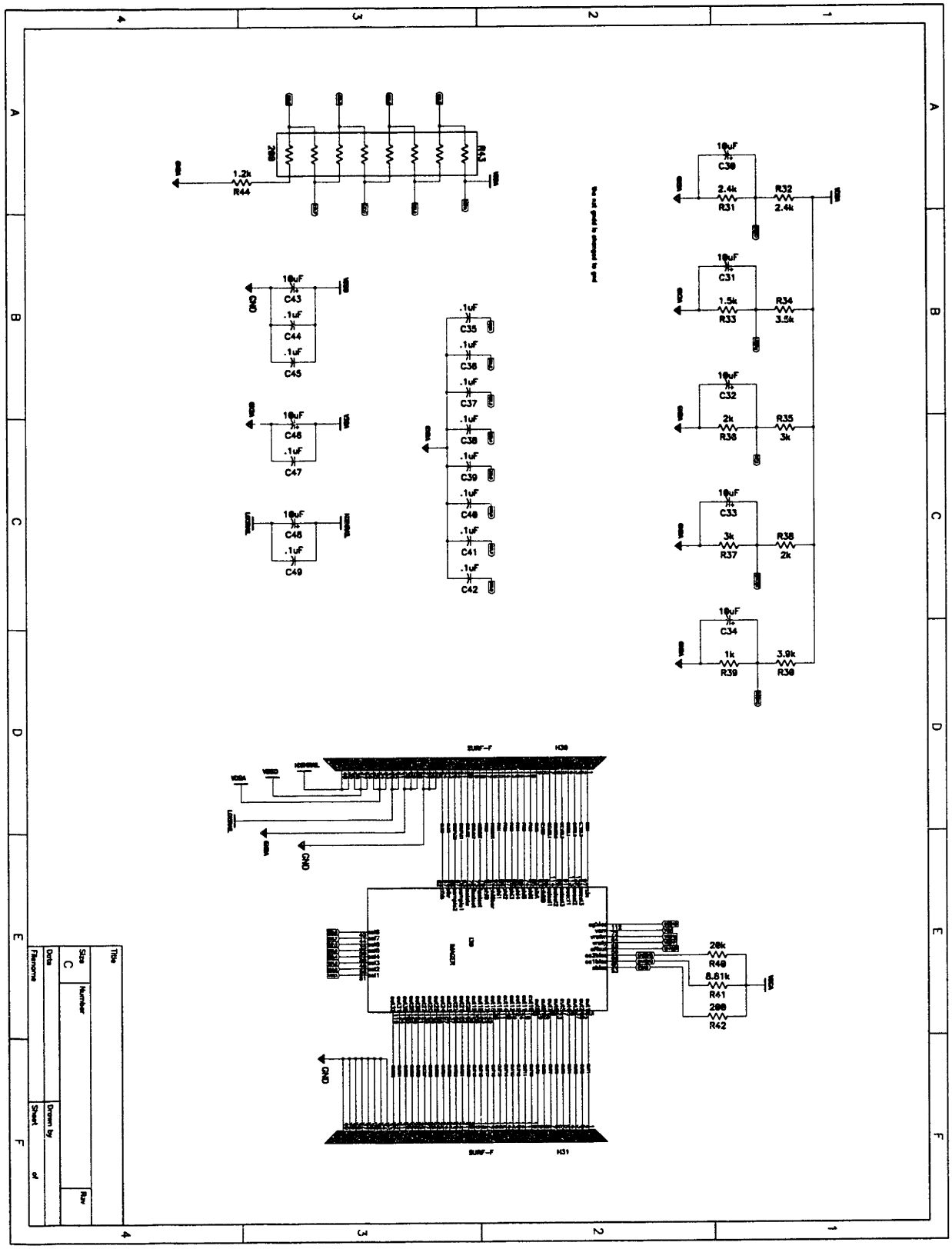

Figure B.1: Analog components and socket for imager.

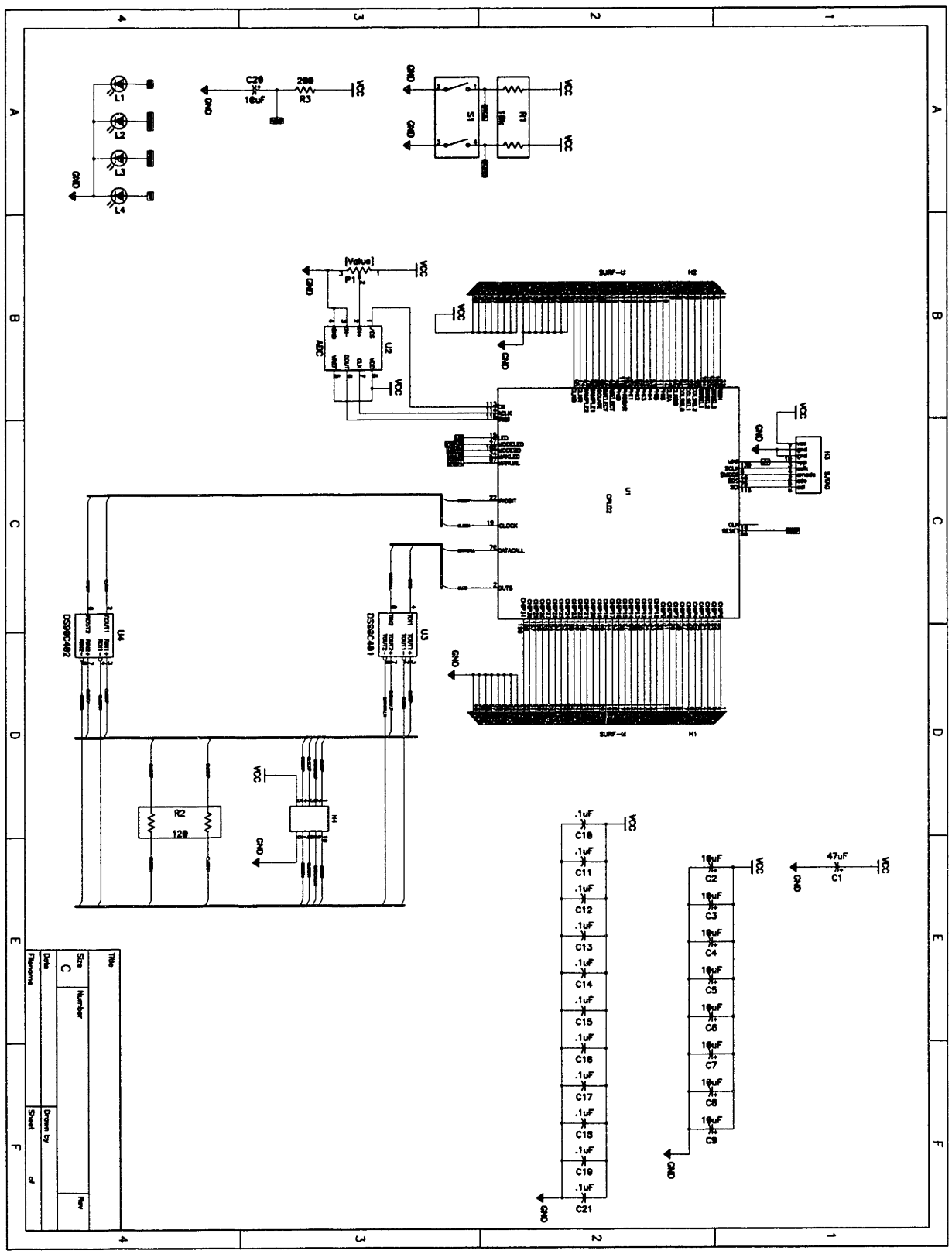

Figure B.2: Digital controller and interface for receiver.

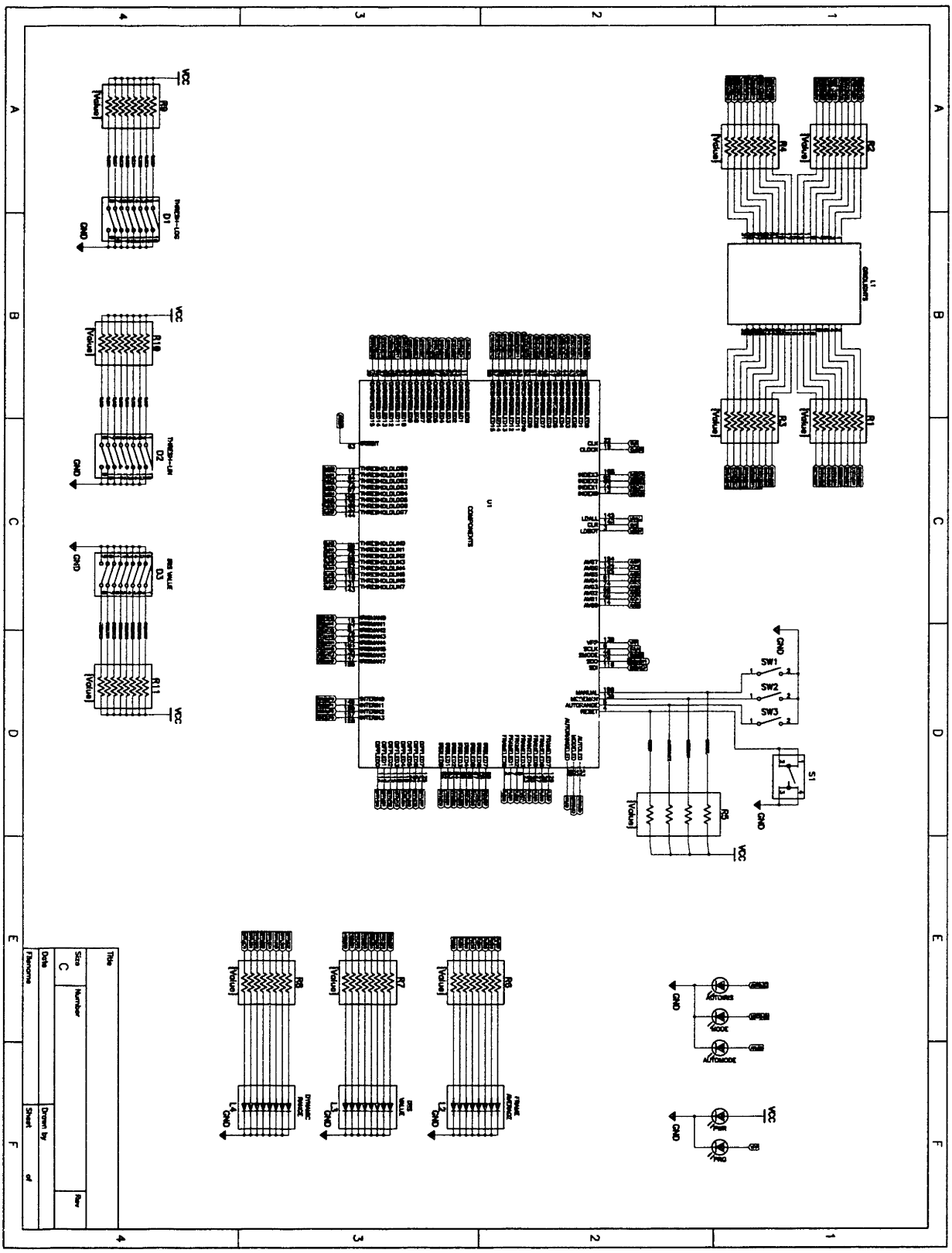

Figure B.3: Controller and comparator for automatic brightness algorithm.

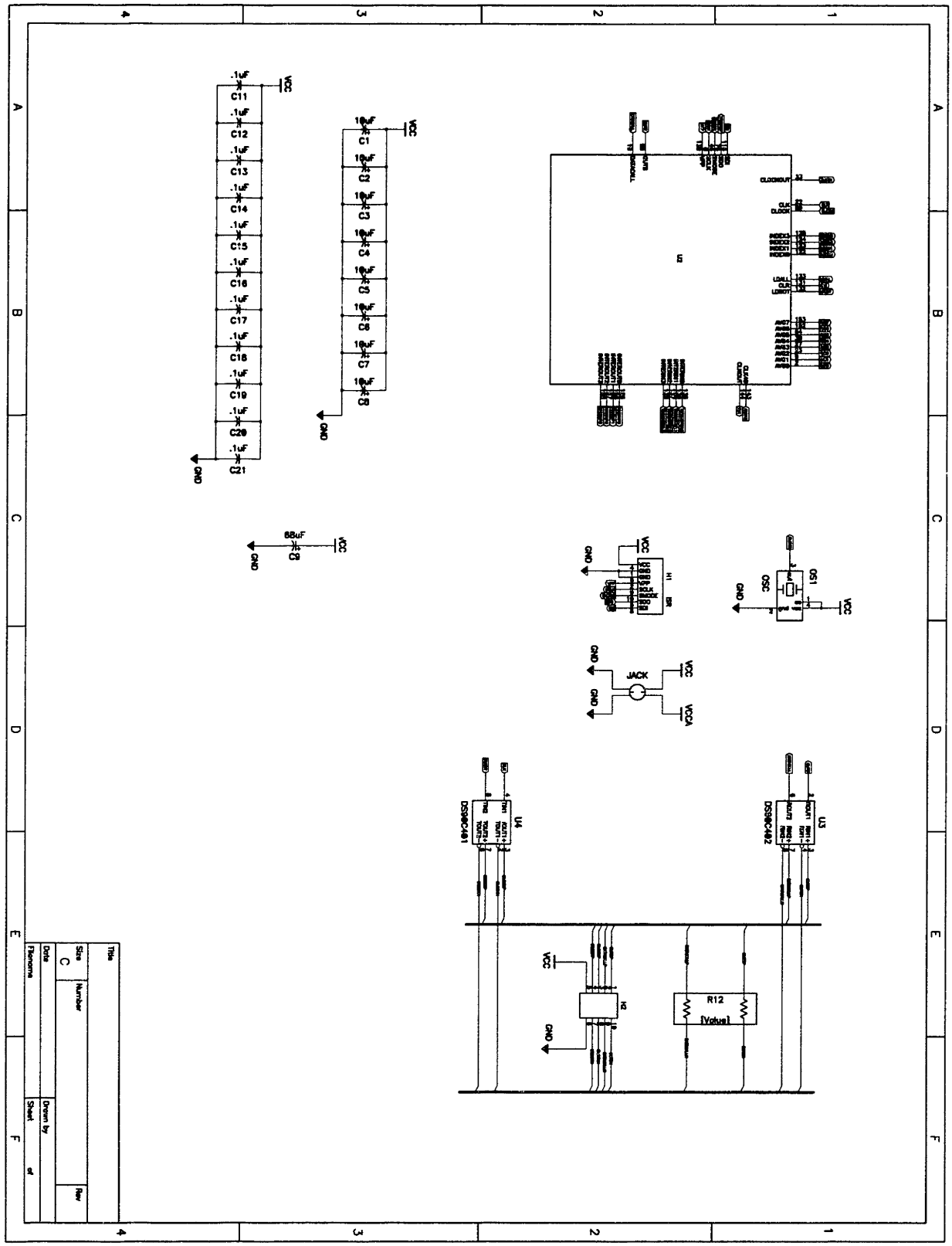

Figure B.4: Controller and programming interface for autobright.

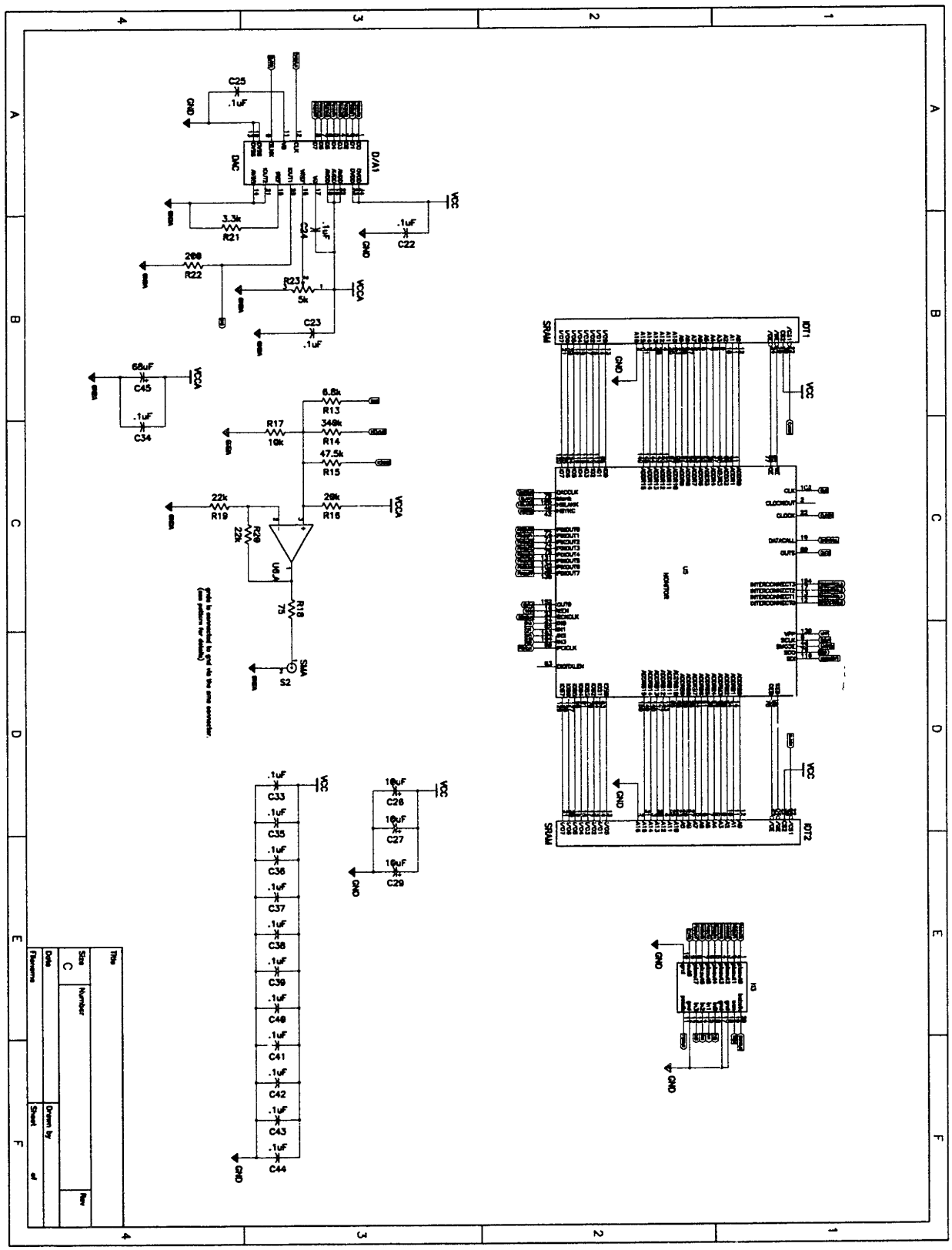

Figure B.5: NTSC encoder and format converter for display.

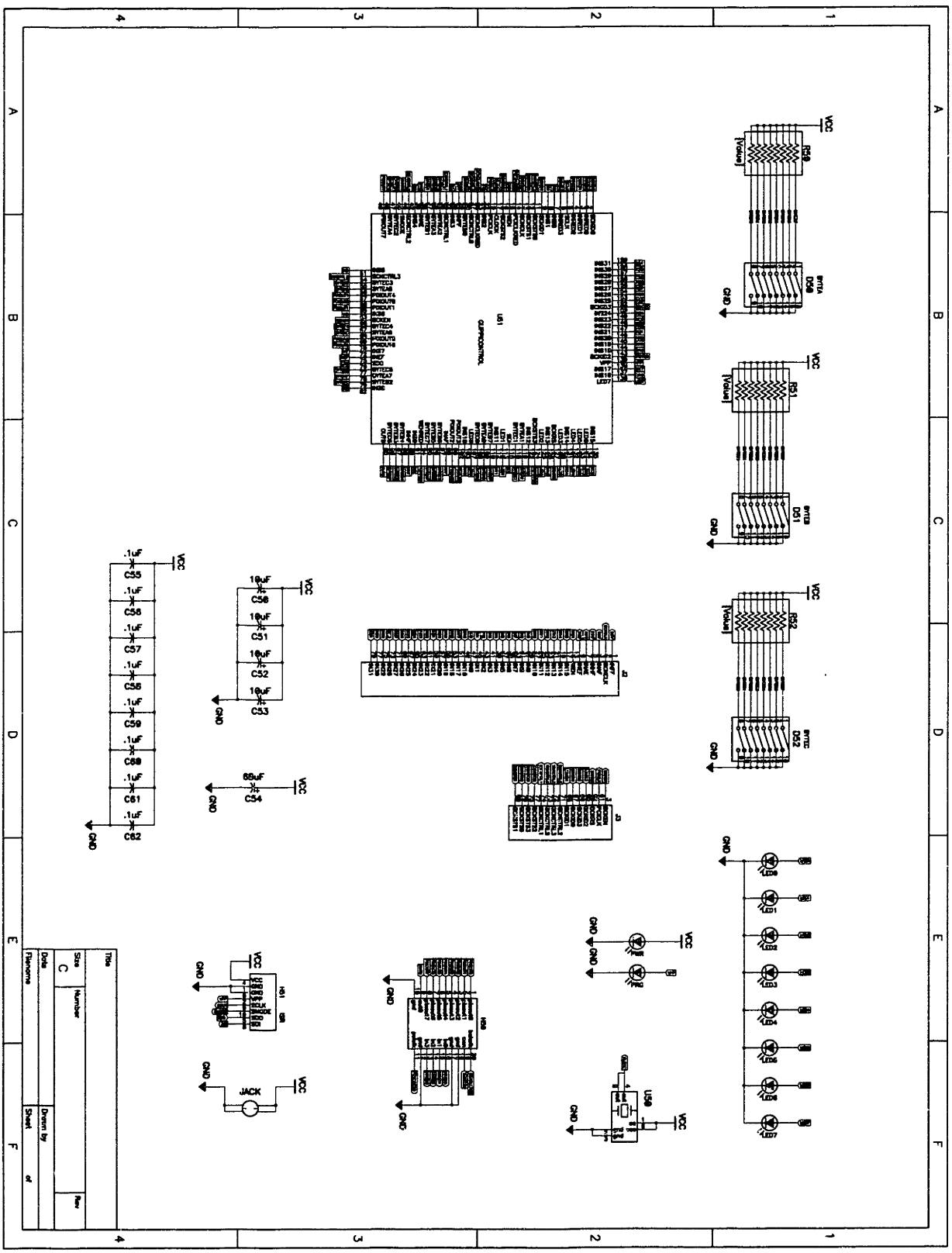

Figure B.6: Tadpole daughter card for GuPPI.

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ COPIES: (Archives) Aero Dewey (Eng) Hum

Lindgren Music Rotch Science

TITLE VARIES: ▶ ☐ _____

NAME VARIES: ▶ ☒ Glen

IMPRINT: (COPYRIGHT) _____

▶ COLLATION: 141 P

▶ ADD: DEGREE: S.B. ▶ DEPT.: E.E.

SUPERVISORS: _____

NOTES:

cat'r: date:

page:

▶ DEPT: E.E. ▶ J38+1416

▶ YEAR: 1999 ▶ DEGREE: M.Eng.

▶ NAME: FIFE, Keith G.