1999

A multiple in-camera processing system for machine vision.

Roberto. Muscedere
University of Windsor

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A Multiple In-Camera Processing System for Machine Vision

by

Roberto Muscedere

A Thesis
Submitted to the College of Graduate Studies and Research through the Department of Computer and Electrical Engineering in partial fulfillment of the requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada

September 1999

Approved By:

Dr. A. Jaekel (External examiner)

Dr. M. Ahmadi (Internal Reader)

Dr. J. J. Soltis (Internal Reader)

Dr. G.A. Jullien (Supervisor)

Dr. N. Biswas (Chair)

Abstract

In a typical machine vision application, a line-scan camera positioned on the production line captures images of the parts to be inspected and sends them to the machine vision computer. The computer then uses high-speed data acquisition devices and sophisticated analysis software to extract information from these cameras and generates decisions about the product and manufacturing system. As the manufacturing systems increasingly generate more fine featured and advanced products, the need for higher resolution and faster processing of these camera images is necessary to maintain quality control.

To reduce the overwhelming amount of data from multiple camera systems to the analysis computer, an in-camera processing system is introduced. This system involves placing a computing system inside the camera which can perform similar operations to the analysis system, but without all of the additional overhead components.

The work presented in this thesis describes an enhanced embedded system which is mounted into a DALSA line-scan camera. This system provides support for real-time one dimensional signal processing with the aid of integrated hardware and software resources.

Acknowledgments

There are several people who deserve my sincere thanks for their generous contributions to this project.

I would first like to thank my supervisor Dr. G. A. Jullien for his guidance, advice and for bringing this challenging project to my attention. I am grateful to Dalsa Inc. for providing funding, the camera systems, privileged technical information and their services in assembling the PCBs. I would also like to thank Hossain Hajimowlana for creating algorithms which demonstrated this projects capabilities, and my committee members Dr. Arunita Jaekel, Dr. Majid Ahmadi and Dr. James Soltis.

I would also like to recognize the following individuals and corporations for their contributions: Marjan Shahkarami for her time and comments on the first drafts of this thesis, CMC for providing and supporting the design software and computing hardware which made this project possible, Micronet R&D for providing financial and networking support, Robert Mavrinac for donating additional computing and networking resources, Joe Novosad for his input on practical PCB designing, and Bruce Watt for his help in implementing post-design changes.

I would also like to thank Electrosonic, Molex, Samsung, Texas Instruments and Xilinx for their kind donations of software, devices and components used in this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ALU	*Arithmetic Logic Unit*
API	*Application Program Interface*
AR	*Auto Regressive*
ATF	*Asynchronous Transmit FIFO*
ATM	*Asynchronous Transfer Mode*
bps	*Bits Per Second*
Bps	*Bytes Per Second*
BUFFALO	*Bit User's Fast Friendly Aid to Logical Operation*
CCD	*Charge Coupled Device*
CLB	*Configurable Logic Block*
CMOS	*Complimentary Metal Oxide Semiconductor*
CPLD	*Complex Programmable Logic Device*
CRC	*Cyclic Redundancy Check*
CSR	*Control and Status Registers*
DARAM	*Dual Access Random Access Memory*
DFF	*Data Flip Flop*
DLL	*Dynamic Link Library*
DPRAM	*Dual Port Random Access Memory*
DSP	*Digital Signal Processor*
EEPROM	*Electrically Erasable Programmable Read Only Memory*
EPROM	*Electrically Programmable Read Only Memory*
FB	*Function Blocks*
FIFO	*First In First Out*
FPGA	*Field Programmable Gate Array*

GRF	*General Receive FIFO*
GUI	*Graphical User Interface*
I/O	*Input/Output*
IBM	*International Business Machines*
IEEE	*Institute of Electrical and Electronics Engineers*
IOB	*Input/Output Block*
ITF	*Isochronous Transmit FIFO*
JTAG	*Joint Test Action Group*
LLC	*Link Layer Controller*
MAC	*Multiply and Accumulate*
MCU	*Microcontroller Unit*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PCI	*Peripheral Component Interconnect*
PHY	*Physical Interface*
PLCC	*Plastic Lead Chip Carrier*
PLL	*Phase Locked Loop*
RAM	*Random Access Memory*
RLE	*Run Length Encoding*
ROM	*Read Only Memory*
SCSI	*Small Computer Simple Interface*
SRAM	*Static Random Access Memory*
TI	*Texas Instruments*
TDI	*Time Delay Integration*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
VLIW	*Very Long Instruction Word*

Chapter 1

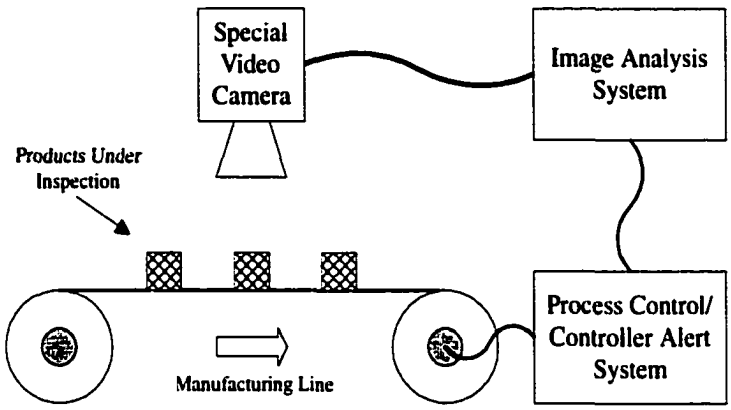
Introduction

1.1 Introduction

Human vision has played an indispensable role in the process of manufacturing products since the beginning of the Industrial Revolution. Human eyes did what no machines could do themselves by locating and positioning work, tracking the flow of parts, and inspecting output for quality and consistency. The requirements today of many manufacturing processes have surpassed the limits of human eyesight. Manufactured items often are produced too quickly or with tolerances too small to be analyzed by the human eye. In response to manufacturers' needs, a new technology known as "machine vision" emerged, providing manufacturing equipment with the gift of sight.

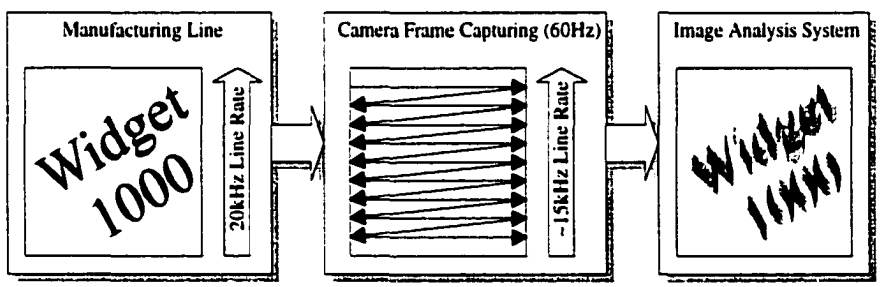
In a typical machine vision application, a special video camera positioned on the production line captures images of the parts to be inspected and sends them to the machine vision computer. The computer then uses sophisticated analysis software to extract information from these digital images and generates decisions about the product manufacturing system (see Figure 1.1).

Figure 1.1 Typical Machine Vision Application



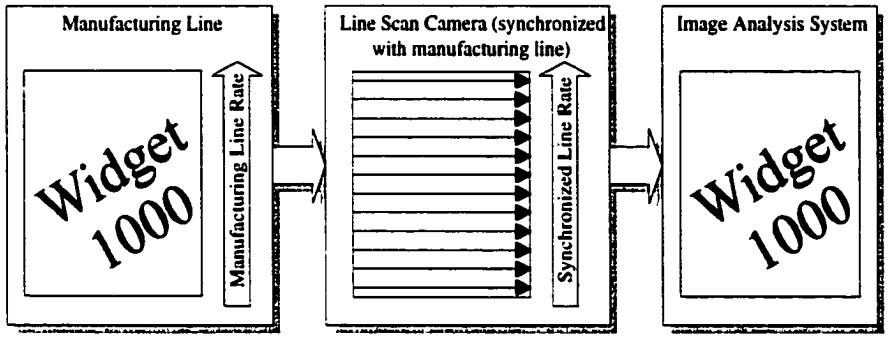
For a moving production line, a standard frame camera (operating at NTSC 60Hz, ~15kHz line rate) is not capable of capturing true images since there is no synchronization between the camera and the manufacturing line (see Figure 1.2). Even if their speeds are matched, there still will be some losses causing image distortion due to the nature of the frame camera.

Figure 1.2 Capturing an Image from a Non-Synchronized Production Line



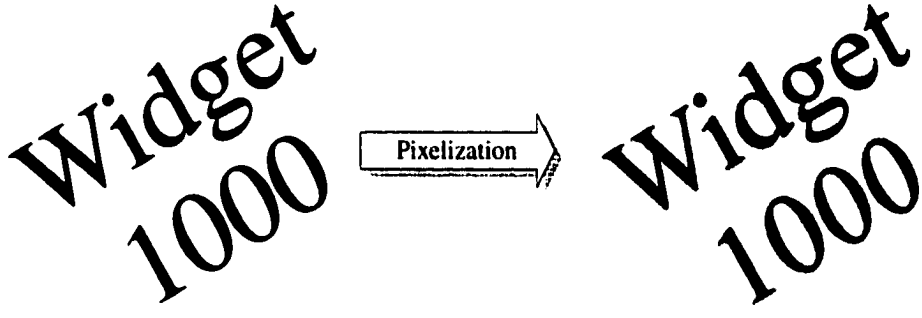
A special continuously capturing camera (line scan) that is synchronized to the motion of the manufacturing line will compensate for the image quality loss of a standard frame camera (see Figure 1.3).

Figure 1.3 Capturing an Image from a Synchronized Production Line



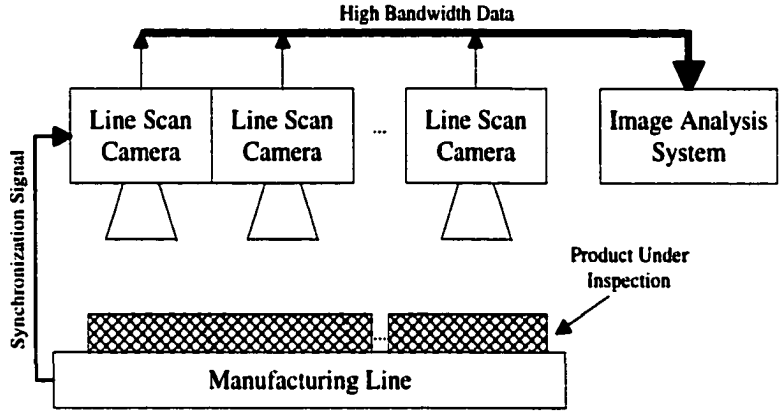
To obtain an accurate digital representation of the image, the camera must be placed an appropriate distance away from the manufacturing line. If the camera is placed too far away from the manufacturing line, important data may be lost by the image pixelization effect (see Figure 1.4).

Figure 1.4 Example of Image Pixelization Effect



To eliminate the problem of image pixelization, multiple cameras can be placed side-by-side and their separate images can be combined by the analysis system into a single complete image (see Figure 1.5).

Figure 1.5 Multiple Camera System

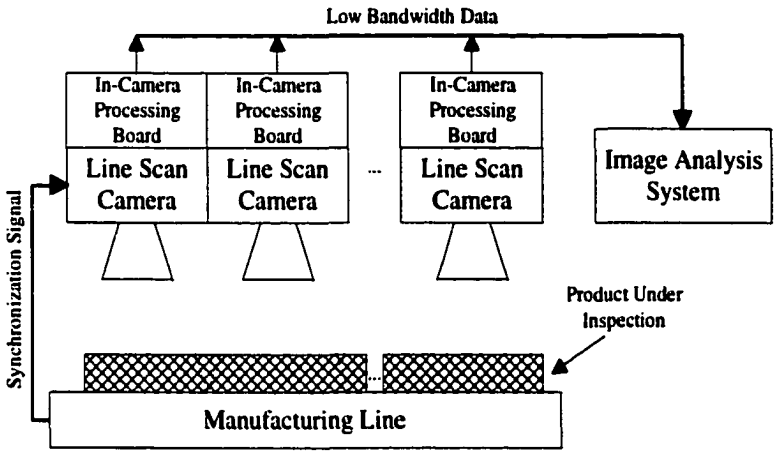


Although multiple cameras solve the pixelization problem, they create a new high bandwidth problem. Similarly as manufacturing system speeds have surpassed the speed of human vision, the computational time for processing very large amounts of image data can overwhelm the image analysis system. To resolve this high bandwidth problem, an in-camera processing board concept is introduced [16].

1.2 In-Camera Processing Board

The concept of the in-camera processing board is simple: Place a computing system inside the camera which can perform similar operations to the analysis system, but without all of the additional overhead components (large operating system, graphical user interface, hard disk, video support, etc.). This in-camera system will then reduce the outgoing bandwidth to the analysis system so that it can handle data from multiple cameras (see Figure 1.6).

Figure 1.6 In-Camera Processing Board Reduces Bandwidth to Analysis System



Due to PCB space limitations inside the cameras, the early in-camera system used a single Field Programmable Gate Array (FPGA) to perform real-time processing directly on the camera's digital video stream. Such a system is limited in its processing capabilities and is unable to perform the higher level image processing required for two dimensional image data analysis.

1.3 First Generation System

A previous research project used a design based on the above system to transmit the processed data from the FPGA over a lower bandwidth bi-directional medium. The permanently programmed FPGA is replaced with an in-system reprogrammable FPGA to allow the processing algorithms to be easily changed. This new system also included a FIFO (simulated by a RAM device) and a microcontroller unit (MCU) to facilitate system control and the bi-directional communication [15].

1.4 Thesis Overview

The work performed for this dissertation was the design of a second generation in-camera processing system which improved upon the first generation system described above. This new system also includes a Digital Signal Processor (DSP) for enhanced arithmetic and

high-speed networking to greatly improve communication to the analysis system and other cameras.

1.5 Thesis Organization

This thesis is organized as follows: Chapter 2 provides information of the first generation system designed prior to this project work. It forms the basis for the decisions made in the second generation system. Chapter 3 introduces the reasoning behind the selection of the devices and components for the second generation system. It also covers the basic connectivity issues in enabling the systems operation. Chapter 4 lists the design partitioning steps to realize the system design while maintaining the constraints of the camera. Hardware tests to verify the design's proper fabrication are also explained. Chapter 5 examines the hardware, software and video algorithms performances and offers potential system design improvements. Chapter 6 concludes this thesis by making recommendations on a future third generation system.

Chapter 2

First Generation System

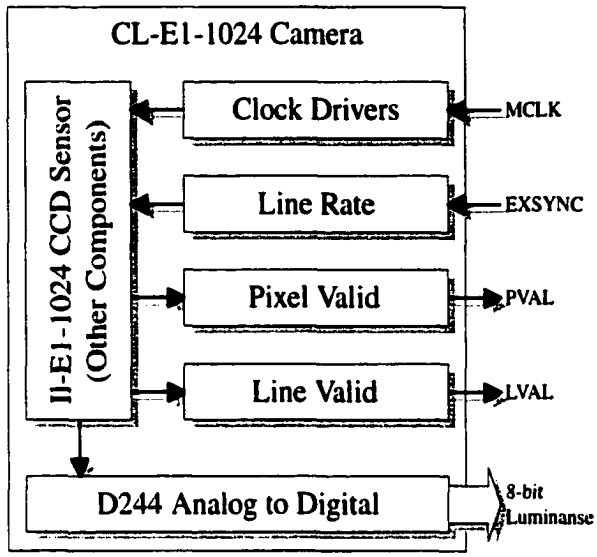
2.1 Introduction

The first generation system was designed prior to the beginning of this work by other students. The state of this system was at the point of initial hardware testing and software design. Before any work could be done on designing the second generation system, the first generation system needed to be evaluated and tested to determine which components of the system would operate within specifications and which systems needed to be modified, improved or removed.

2.2 Camera Specifications

The first generation system was designed for the DALSA CL-E1-1024 line scan camera. This camera operates at a clock speed of 15MHz with an 8 bit luminance image data rate of 7.5MHz and a maximum line rate of approximately 7.5kHz [5]. The clock source of the camera is provided by an external generator and the line rate is controlled by a conveyor or other data rate synchronizing device (see Figure 2.1). The first generation system was physically designed to mount directly into the camera and to connect to the camera's internal signals and buses.

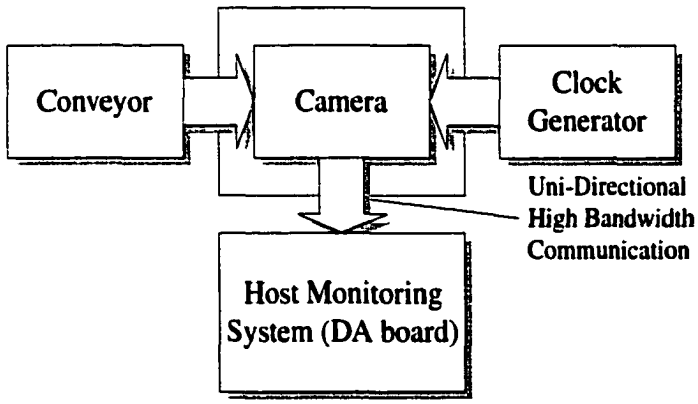
Figure 2.1 CL-E1-1024 System Level Diagram



2.3 Theory of Operation

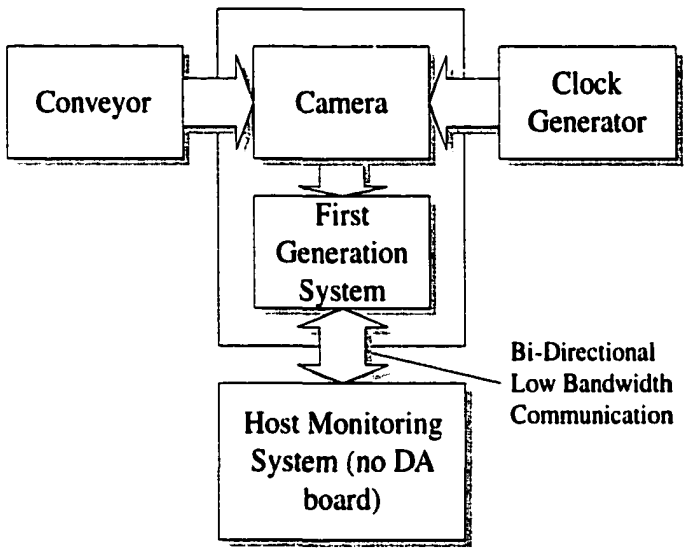
The CL-E1-1024 camera alone has a basic interface. Upon reset, the camera begins transmitting high bandwidth data immediately based on the state of the line valid and clock signals. It is then the responsibility of a digital data acquisition system to process or capture the video data. This is strictly a unidirectional communication relationship; no data from the host ever goes back to the camera (see Figure 2.2).

Figure 2.2 Unmodified High Level System Block Diagram



With the addition of the first generation system, the behaviour of the camera interface changes into a more complex device. The target operation of the system is to reduce the incoming uncompressed data stream from the camera and to transmit it to the host computer over a bidirectional lower bandwidth medium. Although the concept seems simple, it is much more difficult to implement within the systems constraints (see Figure 2.3).

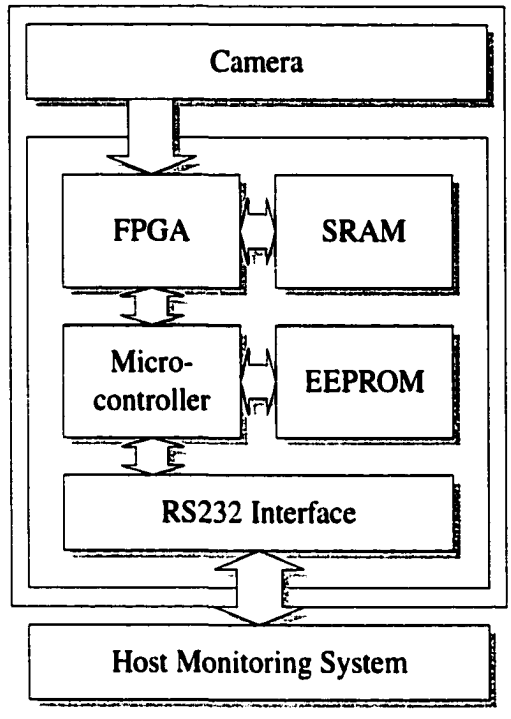
Figure 2.3 First Generation System High Level System Block Diagram



2.4 First Generation System Details

The first generation system consists of a FPGA (to perform real time processing on the digital video stream from the camera), a SRAM (intended for use as a FIFO to compensate for large data bursts of the processed data), a microcontroller (the main controlling unit of the system), an EEPROM (to store microcontroller code and FPGA bitstreams) and a voltage level converter for RS-232 communication to the host monitoring system (see Figure 2.4).

Figure 2.4 First Generation System Component Level Diagram

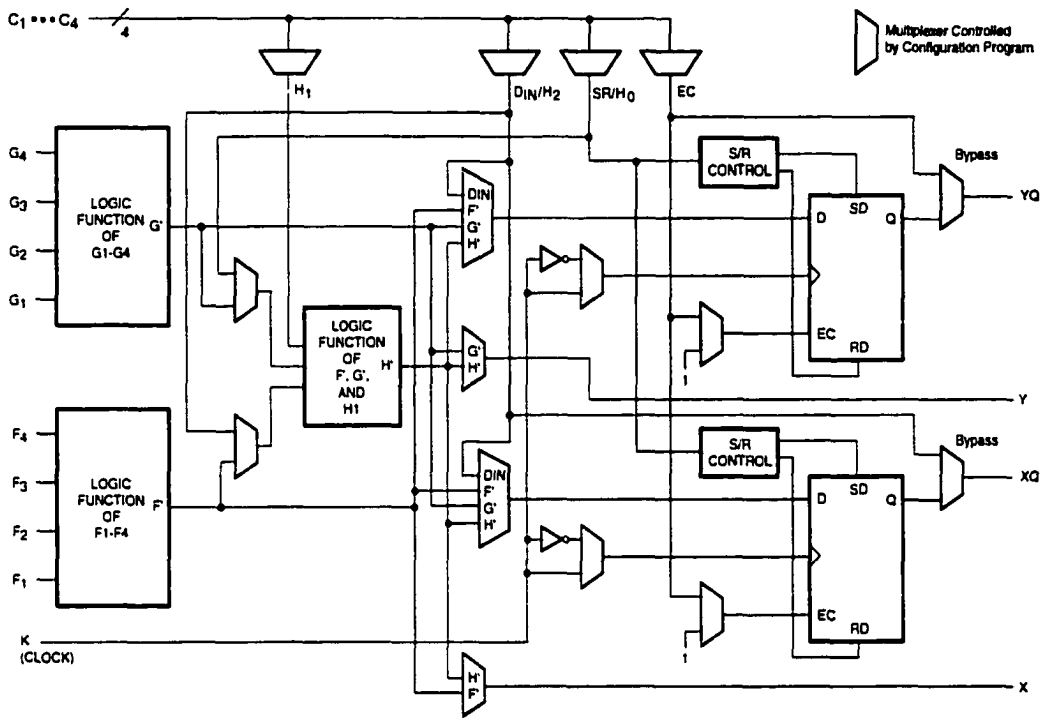


2.4.1 Xilinx 4000E Series FPGA

The Xilinx Field Programmable Gate Arrays (FPGAs) are high-performance, high-capacity SRAM based devices which require programming after reset or power up. Other FPGA manufacturers (such as Altera) produce PROM based FPGAs which can be programmed only once. The Xilinx FPGAs have a long programming life cycle and are ideal for this particular application where the FPGA is intended to be programming for different algorithms. The 4000 series devices are implemented with a regular, programmable architecture of Configurable Logic Blocks (CLBs), interconnected by a hierarchy of routing resources, and surrounded by a perimeter of programmable Input/ Output Blocks (IOBs). Each 4000E series CLB (see Figure 2.5) contains two 4-bit function generators (16x1 lookup tables) with fast carry logic, two clocked (edge or level triggered) "D" flip flops and several multiplexer options which can allow for routing very dense designs. The 4000E series offers the ability of using the two function generators as true RAM elements. Each function generator can be used as a 4-bit dual port RAM or both can be merged into a 5-bit single port RAM. This feature allows each CLBs to contain up

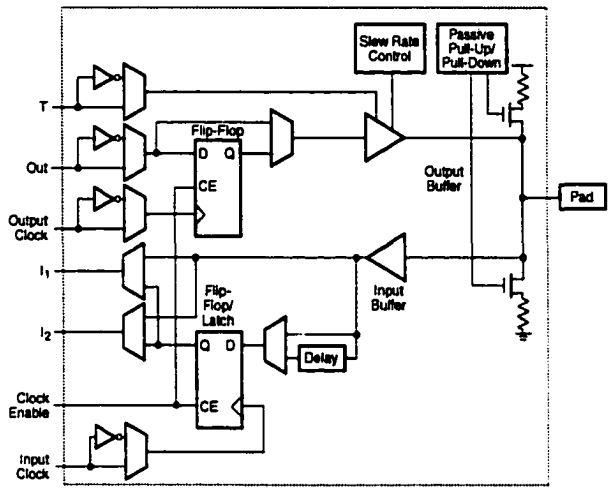
to 34 bits of RAM (only 2 bits per CLB in the 4000 series) thus enabling the ability to realize more complex designs in the FPGA.

Figure 2.5 Xilinx 4000E Series CLB



Each 4000E IOB contains two edge triggered "D" flip-flops with enable lines (one for input and one for output), tri-state control and pull-up and pull-down resistors (see Figure 2.6). This is significantly different from the 4000 predecessor which had no resistors or flip-flop enable functions which had to be implemented in the CLBs.

Figure 2.6 Xilinx 4000E Series IOB



Originally, the first generation system used a Xilinx 4003A-6 FPGA. It was decided to upgrade to a faster 4000E series device since the original was found to be too slow (6ns CLB delay) to handle the 15MHz clock rate and it did not have the appropriate elements to allow for efficient implementation of RAM elements.

2.4.2 Motorola M68HC11E9 Microcontroller Unit

The M68HC11 E series of 8-bit microcontroller units (MCU) combine the M68HC11 CPU with on-chip peripherals. The E series is comprised of many devices with various configurations of RAM, ROM or EPROM, and EEPROM. The particular device used here in this design is the E9 series in a 68 pin PLCC package which contains 12K of ROM (internal BUFFALO monitor for interactive debugging) 512 bytes of EEPROM (programmed code may be executed by internal bootstrap) and 512 bytes of RAM. When in single-chip mode, this device has 40 I/O pins (of differing capabilities) available to the designer.

The MCU is the heart of the first generation system as it is responsible for the operation of the whole embedded system. Its tasks include programming the FPGA, transferring data to and from the EEPROM, and to relay data from the host to the FPGA and vice versa.

System To Host Transmission

The MCU is connected to the host system by means of an RS-232 link. This link is chosen due to its large noise margin (-12V to 12V) which is not easily effected by electrically noisy environments and long transmission distances. The MCU itself outputs and expects TTL signals. An RS-232 voltage level converter is added to the system to transparently adjust for these necessary voltage changes. The speed of this link is chosen to be 9600 baud which is determined by a baud rate generator that divides the MCU's 8MHz crystal oscillator by 13 (see Table 2.1). Since the baud rate is dependent on the driving oscillator, various transmission speeds can be obtained. Most of these speed combinations are not standard and will therefore make it difficult to interface to other standard RS-232 systems such as the ones provided on IBM compatible PCs. Although it is possible to increase the speed of the RS-232 link by lowering the MCU clock speed, this was not done since it would result in a significant overall MCU performance deterioration.

Table 2.1 MCU Baud Rates Related to Crystal Oscillator and Divider

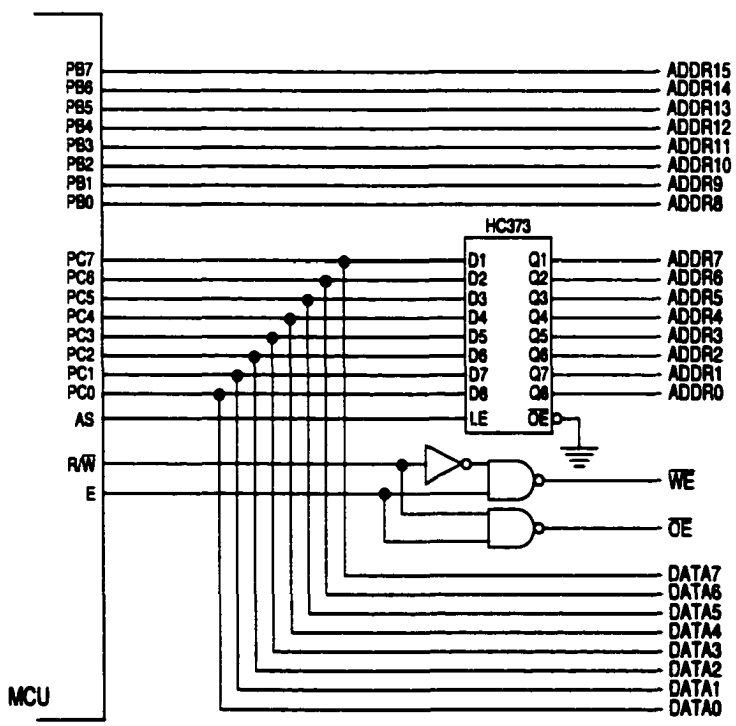
Dividing Factor	Crystal Frequency			
	8MHz	**4.9152MHz**	**4MHz**	**3.6864MHz**
	Highest Baud Rate (standard rates bolded)			
1	125.000K	76.800K	62.500K	**57.600K**
3	41.667K	25.600K	20.833K	**19.200K**
4	31.250K	**19.200K**	15.625K	14.400K
13	**9600**	5.908K	**4800**	4.431K
	2MHz	**1.2288MHz**	**1MHz**	**921.6kHz**
	Bus Frequency			

EEPROM Connectivity

As mentioned above, the MCU is also connected to an external EEPROM device [34]. This device is required to store the FPGA configuration bitstream and any additional MCU code since the RS-232 transmission speeds restrict download speeds from the host to the camera to an unacceptable value. The MCU has the ability to connect to external

memory devices through it's "expanded mode" [14], however, this is not the approach taken here. The expanded mode feature will perform external memory read/writes for memory locations which are not mapped to the internal MCU memory model. This mode also requires several multiplexed I/O pins (address, data, handshaking) in order to address the maximum 64K external memory range. The capacity of the EEPROM in this system is 32K bytes (was originally 16K bytes), and therefore the design does not require the full addressing range and additional external control logic (as seen in Figure 2.7). Instead, an indirect addressing method of connectivity is chosen (see schematic in Appendix A.1 "Schematics" on page 94). By using the user I/O pins available from the MCU (when in single-chip mode), all data, address and control lines are connected; however, the EEPROM access is controlled by setting these I/O signals with specific microcontroller software code.

Figure 2.7 MCU Sample Memory Interface in Expanded Mode



FPGA Connectivity

The MCU connects to the FPGA using a parallel structure. The configuration of the FPGA is set to require a parallel download (determined by 3 pins on the FPGA tied to power/ ground in a particular combination) with hardware handshaking. Along with the handshaking lines are special programming and error signals. With all of these connections and proper software code, the MCU will program the FPGA reliably without any flaws.

2.5 Detailed System Operation

Upon system reset, the MCU boots up by executing the main code within its internal EEPROM memory. This internal EEPROM code also contains a series of routines common to all of the system microcontroller functions. These routines include code that interfaces to the RS-232 communication port and handles the external EEPROM as a virtual disk where all MCU code and FPGA bitstreams can be loaded and saved. As mention previously, the EEPROM is necessary due to the slow speed of the RS-232 link. If the FPGA were to be programmed with data directly from the host computer, the time required would be 15 seconds; a considerable and unacceptable delay.

This main routine initializes all the on-board devices and synchronizes with and waits for commands from the host monitoring system (see Figure 2.8). These commands are completely programmable by the host and range in function from programming the FPGA to upgrading the firmware (see Table 2.2).

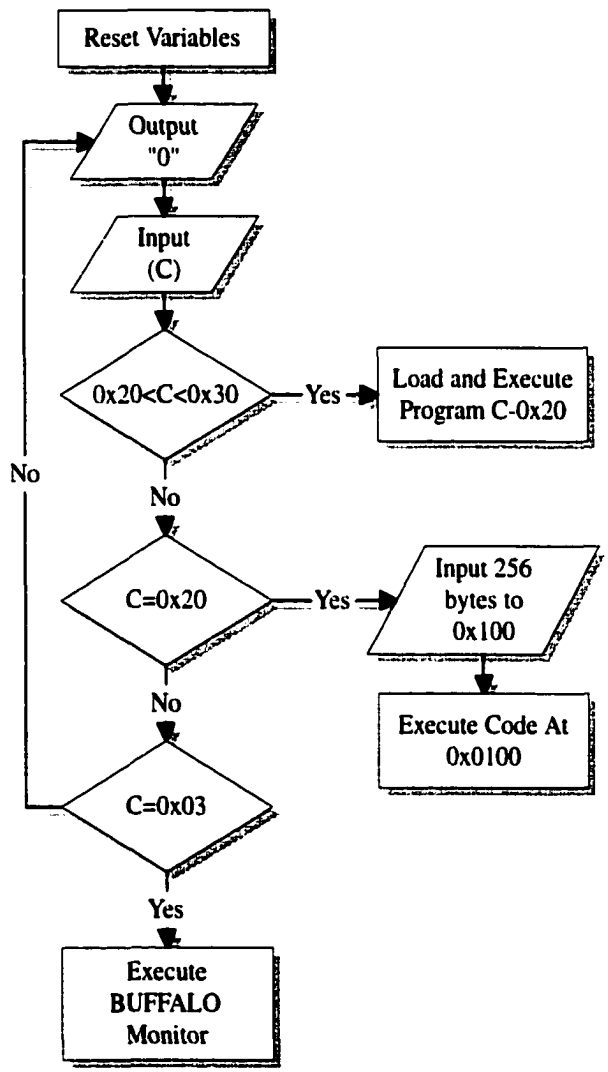
Table 2.2 Top Level System Commands

Command	Explanation
0x20	Executes the following 256 bytes of code at memory location 0x0100
0x21	Download microcontroller code into EEPROM
0x22	Download bit stream from host to EEPROM
0x23	Reset FPGA, download contents of EEPROM bit stream to FPGA and begin capture routine
0x24	Delete microcontroller code from EEPROM

Table 2.2 Top Level System Commands

Command	Explanation
0x25	Download FPGA algorithm properties to EEPROM
0x26	Upload FPGA algorithm properties from EEPROM
0x03	Exits to BUFFALO Monitor for interactive system debugging

Figure 2.8 Detailed Flowchart of Main Routine

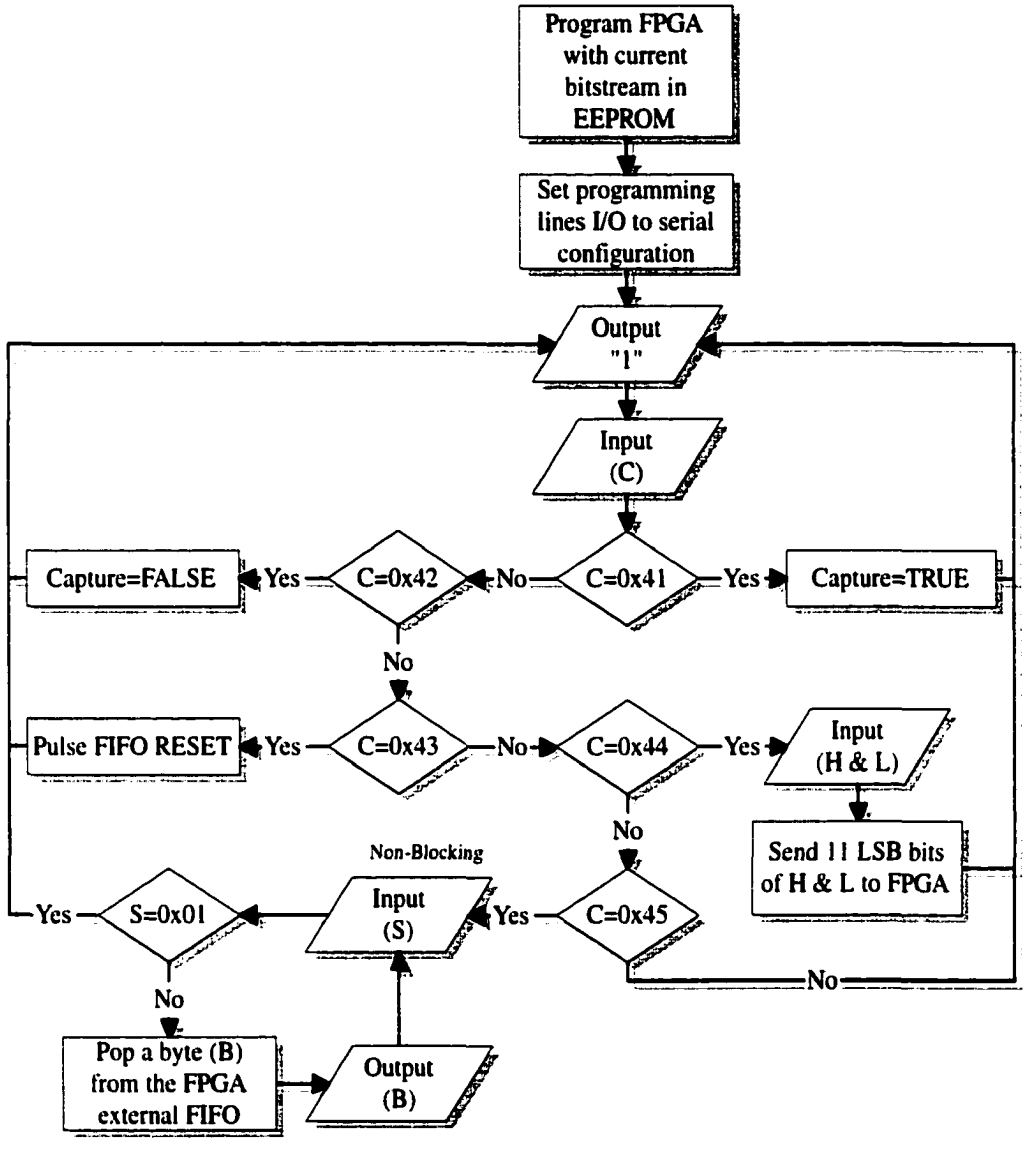


When a valid bitstream is present in the external EEPROM, the processing or capture phase can begin (see Figure 2.9). This portion of the system behaves like the main routine where different commands are executed from the host (see Table 2.3).

Table 2.3 Capture Level Commands

Command	Explanation
0x41	Enable FPGA data processing
0x42	Disable FPGA data processing
0x43	Reset FPGA internal and external FIFOs
0x44	Transmit a parameter to the FPGA
0x45	Begin upload of FIFO's popped contents
0x01	Return to top level

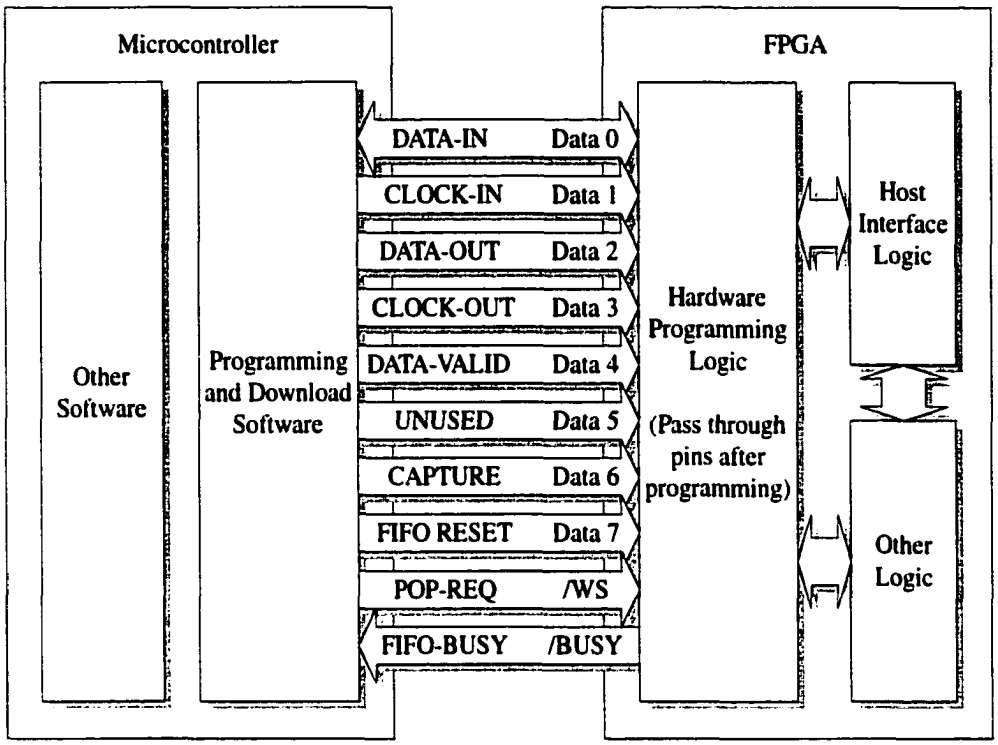
Figure 2.9 Detailed Flowchart of Capture Routine



2.6 Host Communication Interface

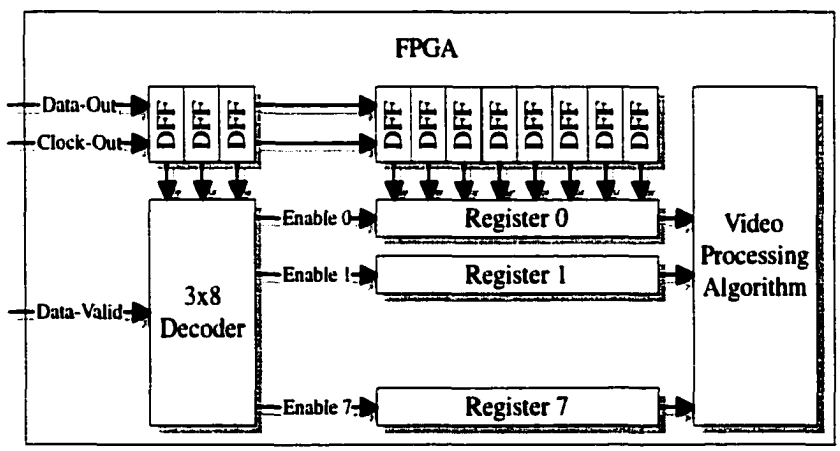
After the microcontroller programs the FPGA by the parallel programming method, the programming lines are then treated as serial communication signals to the FPGA logic. The microcontroller can either send data (programmable algorithm parameters), receive data (popped contents from external FIFO) or read/write data using static logic control lines (see Figure 2.10).

Figure 2.10 Block Diagram of Microcontroller to FPGA Communication



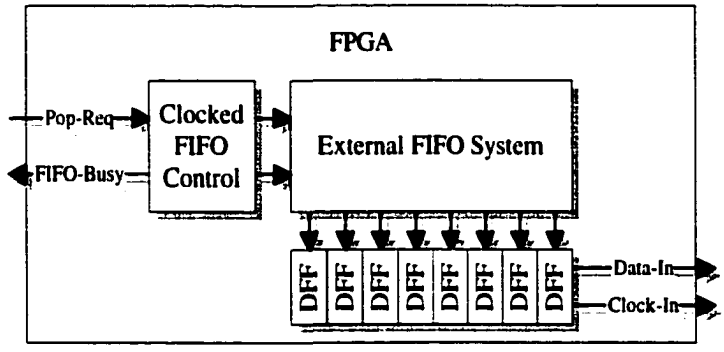
In order to send data from the MCU to the FPGA a simple clocked unidirectional hardware serial communication method is used (shown in Figure 2.11) which requires 3 output signal lines.

Figure 2.11 MCU to FPGA Communication



Receiving data from the FPGA uses a bi-directional hardware serial communication method (shown in Figure 2.12) which requires only 4 signal lines.

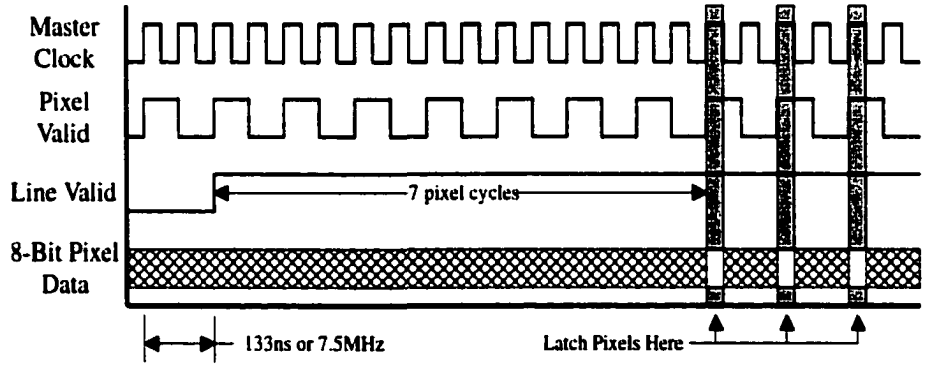
Figure 2.12 FPGA to MCU Communication



2.7 Video Data Processing

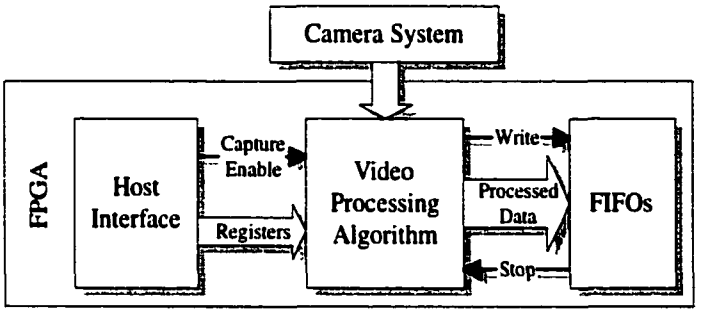
Before any real-time video data processing can be performed in the FPGA, the timing specifications of the camera have to be met in order to obtain the correct data (as shown in Figure 2.13).

Figure 2.13 CL-E1-1024 Output Signals



The FPGA based video logic uses signals from the camera, host interface system (provides programmable parameters) and FIFO system to process the incoming video properly and to transmit it reliably to the host monitoring system (see Figure 2.14).

Figure 2.14 Video Processing Algorithm Connectivity



Any algorithm that adheres to the camera timing specifications and the interface characteristics will operate within this system. The following sections provide information on a few of the simple algorithms implemented in the first generation system to test its operation. The hardware description code for these algorithms can be found in Appendix B.2.4 "Video Processor VHDL Code" on page 162.

2.7.1 Minimum/Maximum Algorithm

The minimum/maximum algorithm simply replaces any video data which falls between a specified range with a background value (see Figure 2.15 and Figure 2.16).

Figure 2.15 Graphical Interpretation of the Minimum/Maximum Algorithm

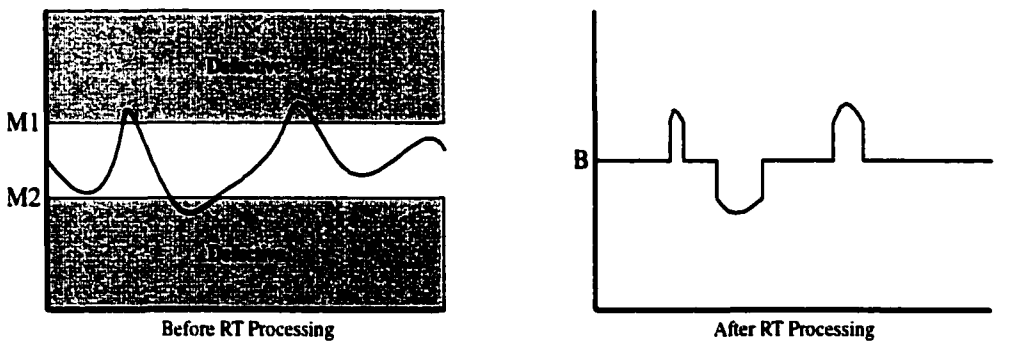
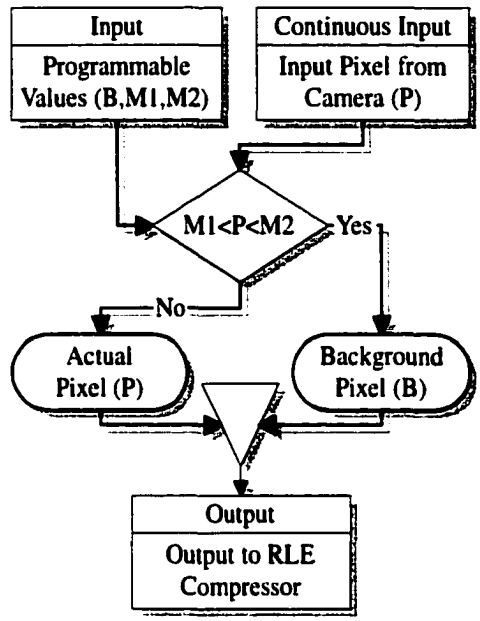


Figure 2.16 Detailed Flowchart of the Minimum/Maximum Algorithm



2.7.2 Range Algorithm

The range algorithm is simply the complement of the minimum/maximum algorithm. Any video data which does not fall between a specified range with be replaced with a background value (see Figure 2.17 and Figure 2.18).

Figure 2.17 Graphical Interpretation of the Range Algorithm

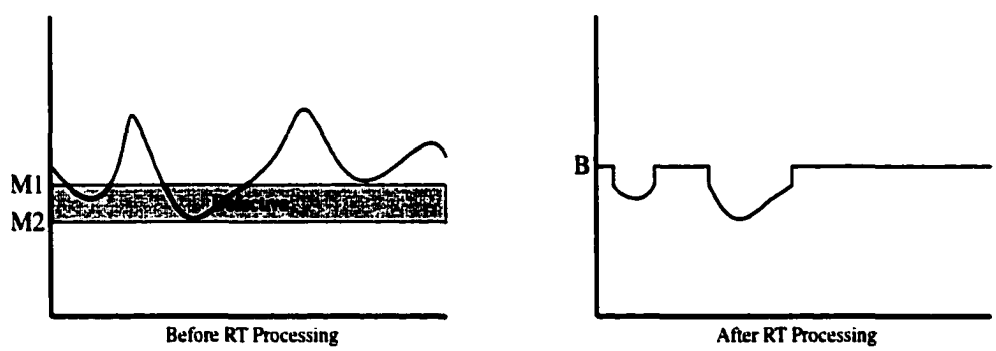
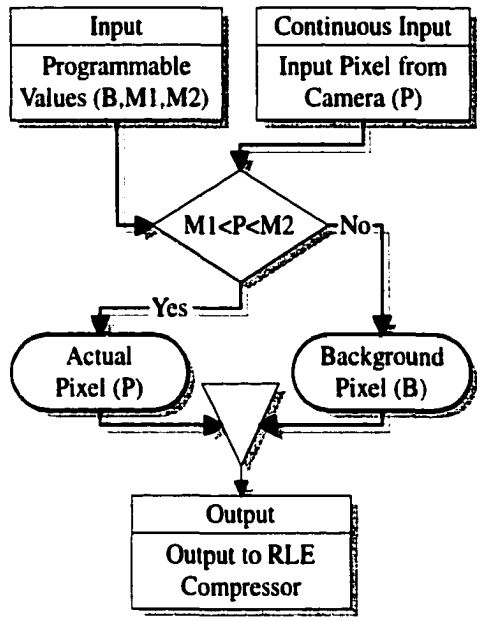


Figure 2.18 Detailed Flowchart of the Range Algorithm



2.7.3 Delta Threshold Algorithm

The delta threshold algorithm calculates the absolute slope between two adjacent values and replaces the second with the background value if that slope is lower than specified (see Figure 2.19 and Figure 2.20).

Figure 2.19 Graphical Interpretation of the Delta Threshold Algorithm

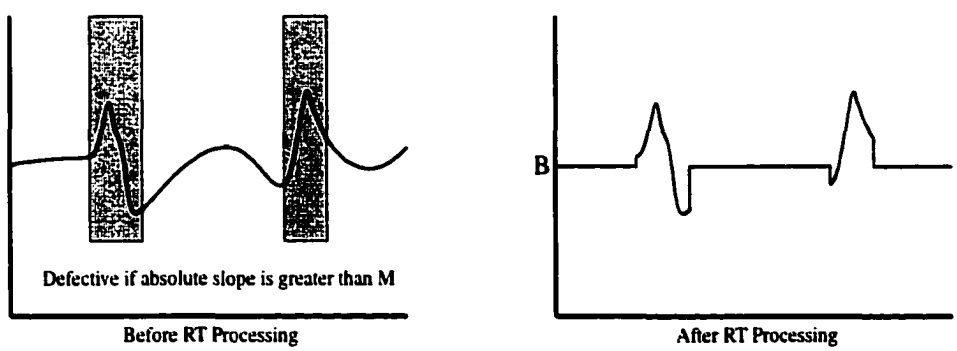
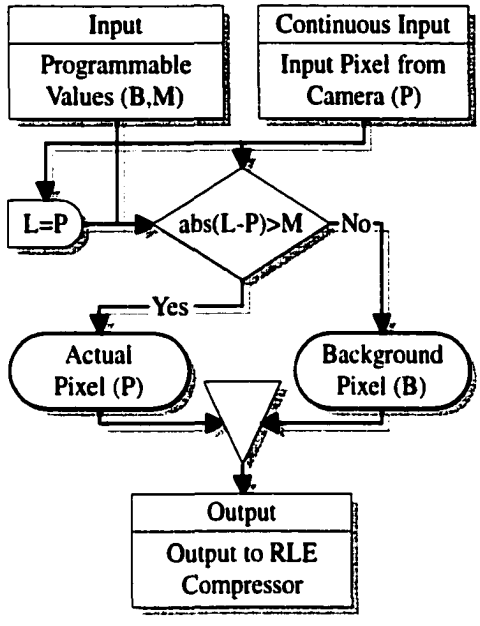


Figure 2.20 Detailed Flowchart of the Delta Threshold Algorithm



2.7.4 Delta Tracker

The delta trackers algorithm operates by tracking the background based on the fluctuations of the incoming values. The pixel value is replaced with the background value when it is within a specified tolerance (see Figure 2.21 and Figure 2.22).

Figure 2.21 Graphical Interpretation of the Delta Tracker Algorithm

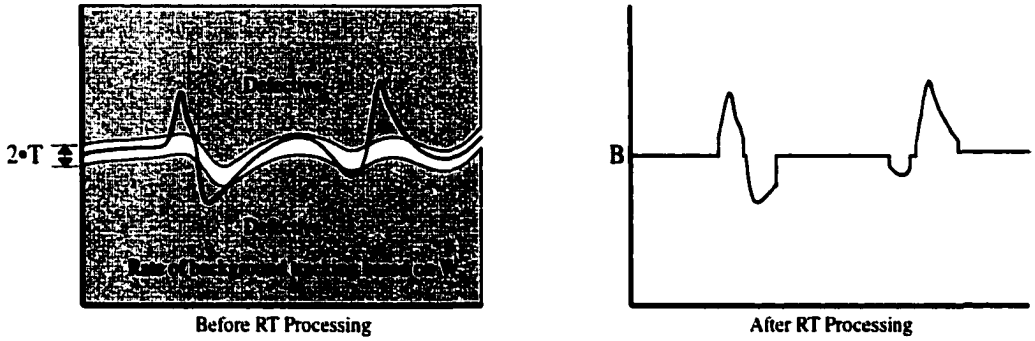
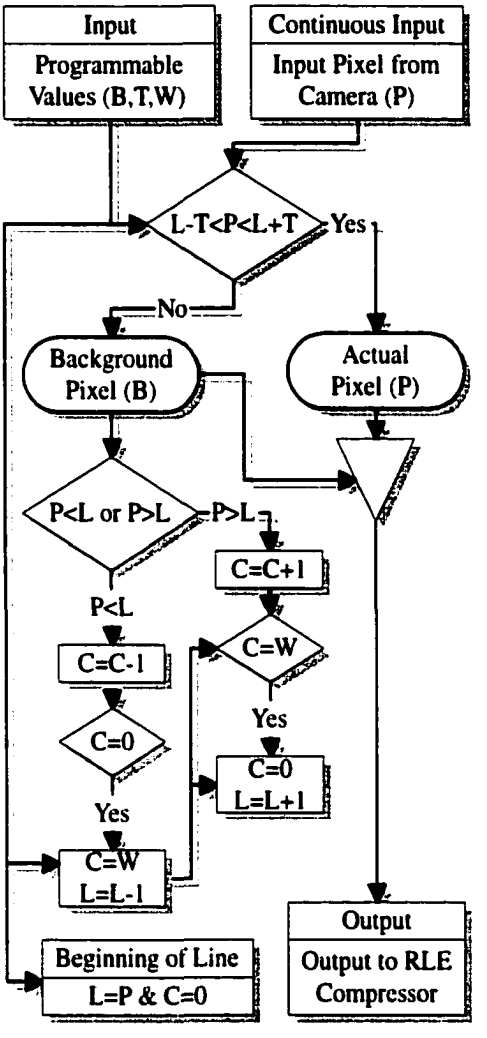


Figure 2.22 Detailed Flowchart of the Delta Tracker Algorithm



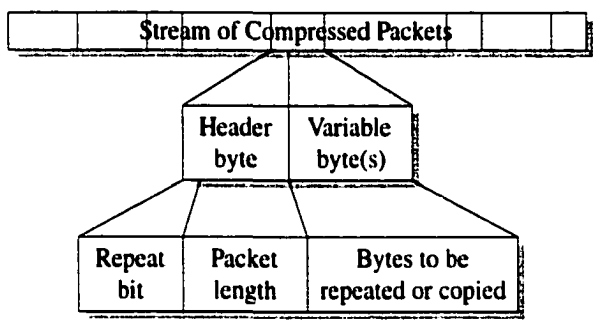
2.8 Data Compression

After the FPGA performs its selective pixel elimination, the new data is sent to the microcontroller which then transmits it to the host. This data must be as compressed as possible since the transmission speed to the host is slow relative to that of the camera bus. At best case, the FPGA algorithm would eliminate all pixels in a line from being defects, therefore this case should transmit the minimal amount of data. Since the FPGA is operating in real time, a data compression scheme must be designed which will compress the data efficiently and use very little resources.

2.8.1 Run-Length Encoding (RLE) Compression

RLE compression is a very simple approach to data compression. It compresses best when there is a long sequence of duplicate bytes; however, continuous non-repeating bytes cause non-optimal compression. Although the RLE algorithm differs from one implementation to another, the concept remains the same. A compressed stream consists of a series of packets, where each packet begins with a header byte and followed by variable data from 1 to 128 bytes (see Figure 2.23).

Figure 2.23 Decomposition of the Compressed Stream



The header byte (separated into a repeat bit and a 7 bit counter value starting at 0) represents the type of packet and the amount of data remaining in the packet. When the repeat bit is set, the counter value contains the number of times the next byte is repeated; This packet is always 2 bytes. The other packet possibility is when the repeat bit is cleared and the counter value contains the number of bytes remaining in the packet to be copied into the uncompressed stream. See Table 2.4 for some RLE compressed data examples.

Table 2.4 Examples of RLE Compressed Byte Streams

Uncompressed Data	Compressed Packet
25252525252525252525	8925
010304060810	05010304060810
8080808080807F7E80818080808080	8580037F7E80818480
101112131415161718191A1B1C1D	0D101112131415161718191A1B1C1D

2.8.2 Real Time RLE Implementation

Although the RLE compression scheme is relatively simple to implement in software, it is much more difficult to implement in real time hardware. In order to implement the non-repetitive portion of the compressor, a large buffer is needed to store the outgoing stream before the header byte. Potential problems involving throughput and buffer overwriting are very likely, particularly when the compressed stream is being transmitted while new data is being processed. Instead of designing around these problems, a modified RLE compression scheme is introduced.

2.8.3 Modified RLE Compression

The target hardware for the RLE compressor is an FPGA where buffer space or RAM is very costly in terms of resources. To resolve this, the compressor repeat mode is always considered on, thus increasing the counter value to 8 bits. This approach effectively decreases the compression ratio for non-repeating data such that 2 bytes of compressed data represents one byte of uncompressed data. Although this modification does not produce the same level of compression for non-repeating data as the original method, it is implementable with limited hardware.

Defects and Non-Defects

In order to describe a defect in the compressed data stream, the luminance value obtained from the camera is encoded with no modifications. However, a non-defect is replaced by a programmable background value. This background value should be set based on the requirements of the software algorithms in the host monitoring system which will later process the compressed stream. Ideally, the compressed low bandwidth data will recreate the image originally obtained from the camera with only the defects visible.

Enhanced Size Encoding

Using the RLE compression method, as described above, we obtain non-optimal results on long streams of repetitive data. This will most likely occur when encoding long background lines (as shown in Figure 2.24).

Figure 2.24 Compressed Single BLACK Line

FF00FF00FF00FF000400

To increase the compression performance of the encoder for these particular instances, a special header code 0xFF is added and the maximum counter value is reduced from 255 to 254 to compensate. This code allows data to be repeated in large streams without having to re-encode the repeating value (see Figure 2.25). This new code can reduce a compressed long line by 4 bytes.

Figure 2.25 Compressed Single BLACK Line Using Enhanced Size Encoding

FF00FFFFFF08

Line Synchronization

With RS-232 transmission, there exists the possibility that the host may incorrectly receive the encoded stream and the decompressor will misinterpret it, losing synchronization with the beginning of the compressed packets (as seen in Figure 2.26).

Figure 2.26 Example of Possible Transmission Corruption

Normal Stream	Corrupted Stream
FF00FFFFFF08FF00FFFFFF08	FF00FFFF8B08FF00FFFFFF08

To avoid this possibility, two 0x00's are added to the encoded stream (synchronization command) before the beginning of a scan line (see Figure 2.27). It is not necessary to have a synchronization command at every line, in fact in the implemented compressor,

synchronization is transmitted every 64 lines without any noticeable failure in the decompressor.

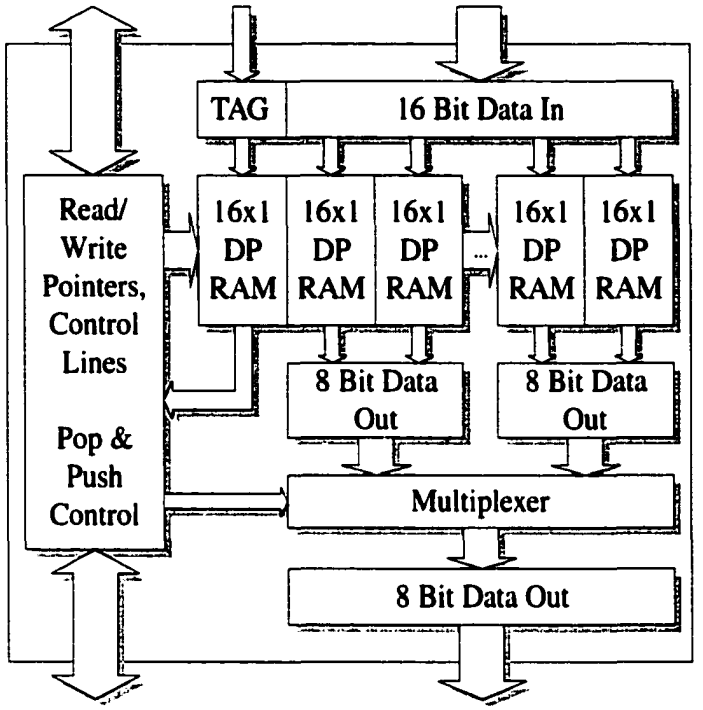
Figure 2.27 Example of Synchronization in a Corrupted Transmission

Corrupted Stream	Corrupted Stream with synchronization
FF00FFFF8B08FF00FFFFFF08	FF00FFFF8B080000FF00FFFFFF080000

2.9 FPGA Internal FIFO

An internal FPGA implemented FIFO is necessary since there is the possibility the external FIFO could be busy servicing a read operation while a write operation is requested. The internal FIFO will hold and postpone the write operations to the external FIFO until it is idle. The internal FIFO also doubles as a data width converter to compensate for the RLE data compressor, which may write 8 or 16 bits to the internal FIFO. This data width is determined by an additional output to the RLE compressor along with the write request line. The external FIFO itself can only handle 8 bit data widths at 15MHz and the compressor can generate up to 16 bits at 7.5MHz. By adding a tag bit to the internal FIFO data input, data may enter as a 16 bit value but leave as two 8 bit values by performing two separate writes to the external FIFO. The internal FIFO is easily realized by using a special CLB dual port RAM component available only in the Xilinx 4000E series FPGAs [35]. Each dual port RAM is a true 16x1 memory device with edge triggered control. A 16-bit by 16 deep FIFO is made by connecting 17 of these RAM components in parallel (as shown in Figure 2.28) and adding logic for pointer control. The hardware description code for the internal FIFO can be found in Appendix B.2.3 "Main VHDL Code" on page 148.

Figure 2.28 Internal FIFO Structure



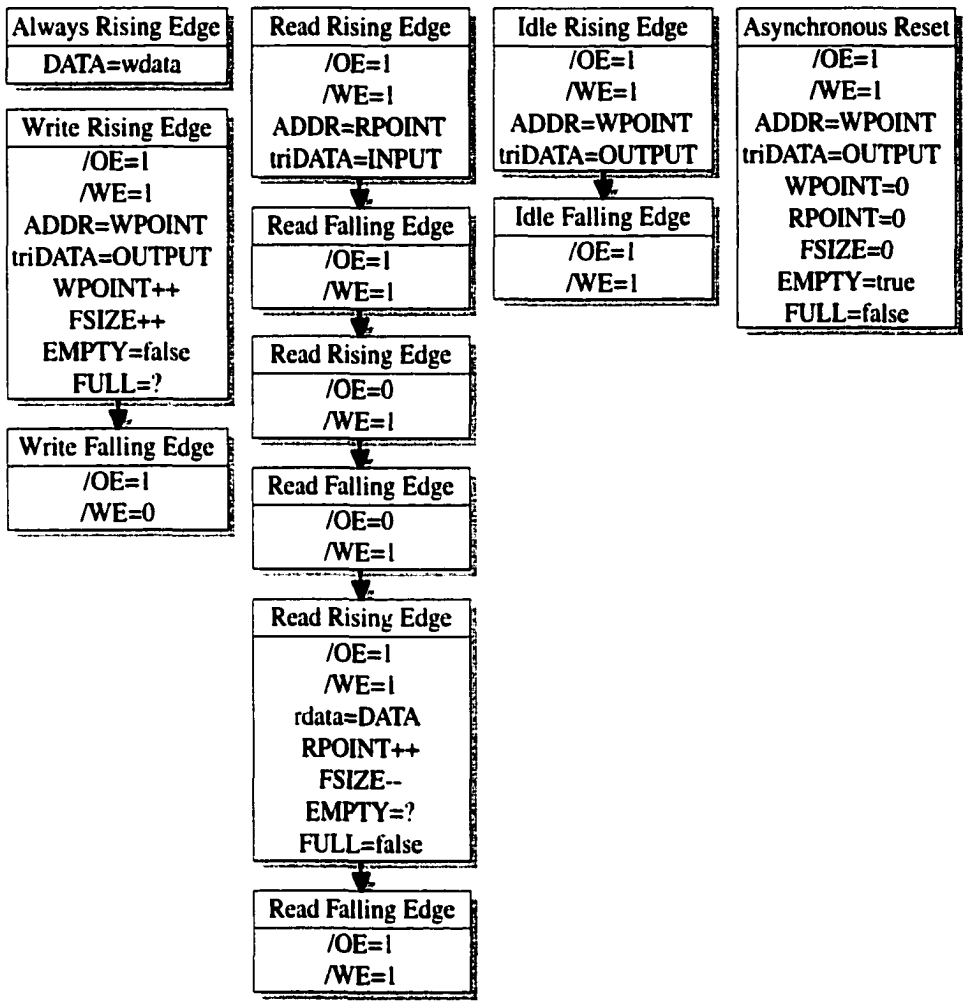
2.10 FPGA External FIFO

Due to the possibility of large data bursts (continuous defects) from the real-time processing algorithms inside the FPGA (up to 15MHz assuming two bytes per defect generated by RLE compressor), the system-to-host transfer rate of 9600 baud is certainly too slow to handle reliable transmission without data overrun. Therefore, a simulated external FIFO is created by a single port SRAM device (128K byte) and additional logic implemented in the FPGA.

The implementation of the external FIFO was difficult due to unexpected read timing problems associated with the particular SRAM device in this system. The SRAM control lines ($\overline{WE}$ and $\overline{OE}$) are controlled by using multi-phase clocking at the camera clock rate of 15MHz. A write operation performed within a single cycle (as expected); however, the read operation did not meet the required one and a half cycle time. The read operations (with back-to-back write) were not proven reliable until the operation was expanded to

three clock cycles. See Figure 2.29 for a complete detail on the cycle operation of the simulated external FIFO.

Figure 2.29 External FIFO Read/Write/Idle/Reset Operations Flow Diagram



2.11 Hardware Design Tools

2.11.1 Synopsys Design Compiler

The Synopsys Design Compiler synthesises VHDL or Verilog code into logical netlists by instantiating a vast library of elements based on the timing models and design constraints of the target hardware. See Appendix B.2.1 "Synopsys Design Compiler Procedure and Scripts" on page 144 for more information.

2.11.2Xilinx XACT

The logical netlist created by Design Compiler is optimized and transformed to the available components of the target FPGA. The software performs automatic placement and routing (if possible) and generates a downloadable bitstream for directly programming the FPGA. See Appendix B.2.2 "Xilinx Scripts" on page 147 for more information.

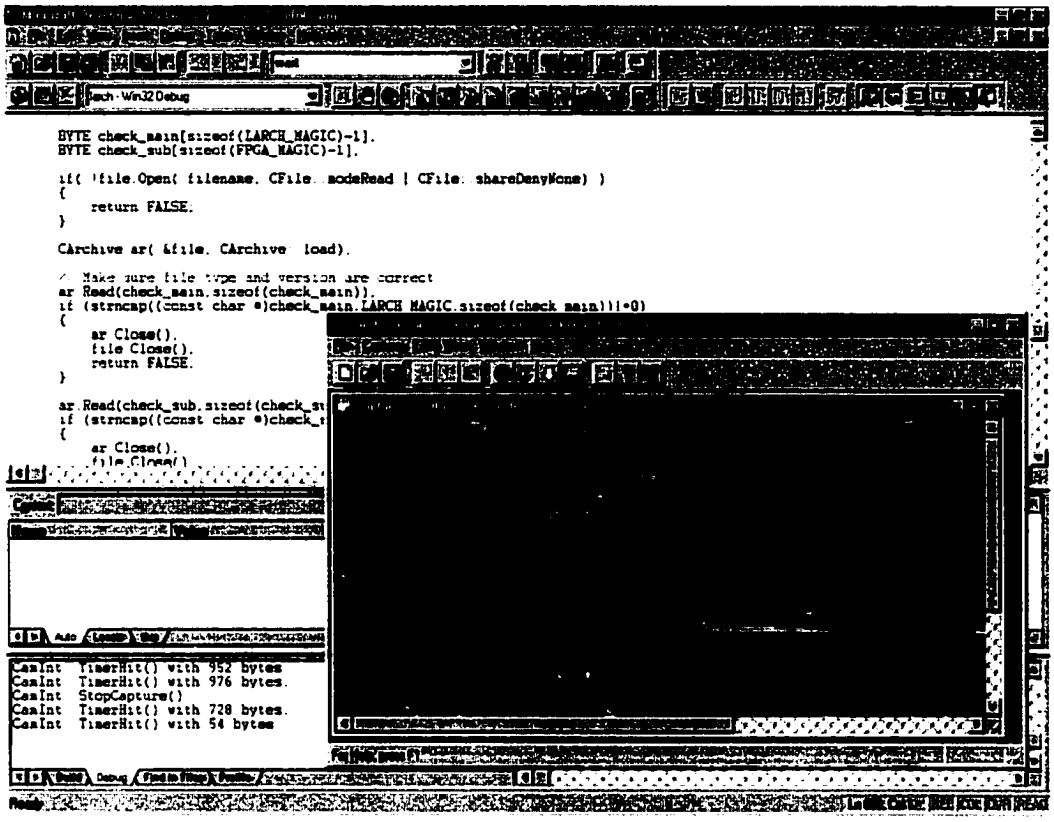
2.11.3Microcontroller Assembler

A freeware mini-assembler provided by Motorola is used to convert text based assembly code [13] into a special format file structure (S19) which is interpreted by the MCU's BUFFALO monitor and the host software. See Appendix B.1.1 "Code Building Utilities" on page 101 for more information.

2.12 Host Software Design

The target host environment is an IBM compatible PC running the Microsoft Windows NT 4.0 which is a 32-bit protected operating system. In order to interface with the RS-232 hardware reliably at a user level for this application, special software coding styles are required to interface with the kernel level. A multi-threaded graphical user interface (GUI) programmed with Microsoft Visual Studio v4 is created (see B.3 "PC Host Software Code" on page 184) which displays the reconstructed video image from the camera incoming data stream (see Figure 2.30).

Figure 2.30 Software Development Environment and Host Monitoring Software



2.13 Summary

The FPGA realtime video processing algorithms and internal FIFO operate as expected at the target 7.5MHz data rate and 15MHz clock rate. However, the external FIFO is not viable in the second generation system since it is unable to be emptied as fast as it can be filled and it simply consumes to many FPGA resources.

The RS-232 transmission is definitely not capable of handling the maximum bandwidth of the processed data from the FPGA video algorithms. The streaming nature and lack of error correction in RS-232 transmission is difficult to compensate for in this system where host-camera synchronization is imperative. An inexpensive synchronous high-speed transmission system which is capable handling network environments with packet structures is more appropriate for this application.

The 68HC11 MCU operates as well as could be expected, however, the 512 byte RAM limitation causes difficulty in coding complex procedures. If the transmission system were to be upgraded, the MCU couldn't handle the additional load. A faster and larger internal memory MCU device is suggested for the second generation system.

Chapter 3

Second Generation System Design

3.1 Introduction

Once the first generation system was fully implemented and tested, several recommendations were made to the design for the second generation system. All decisions for the new system are based on component and device data from the early half of the 1997 year.

3.2 Design Target

The second generation camera is designed to interconnect with the DALSA ML-C3-2048L single channel line scan camera. This camera is capable of operating at 20, 10, 5 and 2.5MHz data rates and clock rates double to that of the data rate. The data rate, analog signal offset, and amplification are programmable through communication to an on-board MCU via an RS232 serial connection [6].

3.3 Design Improvements

3.3.1 Field Programmable Gate Array

Based on the success of the Xilinx FPGA in the first generation system, it was decided to continue using the Xilinx 4000E FPGA series in the 84 pin PLCC package with faster speed grades.

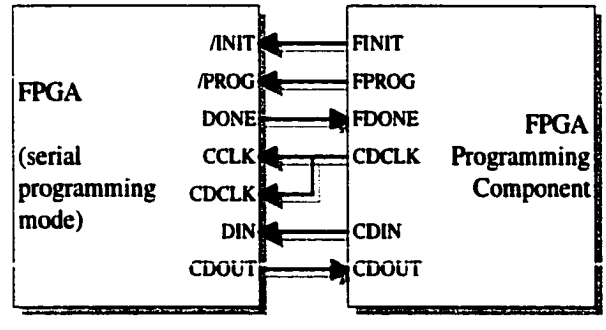
Xilinx 4000EPC84 Series FPGA

The 4000E series were Xilinx's current FPGA product at the time and the performance characteristics of the devices were continually improving. With the 84 pin PLCC package, FPGAs with up to 400 CLBs and half a nanosecond functional block delay are available. Of the 84 pins on the FPGA, 16 are used for power, 5 for serial slave programming, and 64 available for user I/O [38].

Programming Methods

In the first generation system, parallel peripheral mode programming is used which consumes 3 programming pins, 9 user I/O pins and 3 configuration pins. The second generation system uses the serial slave programming method requiring only 4 programming pins and 1 user I/O pin. This user I/O pin is re-used in the real-time function of the camera along with 2 other user I/O pins to facilitate serial communication to the controller (see Figure 3.1).

Figure 3.1 FPGA Serial Slave Interface



User I/O Pins

By freeing up 10 of the FPGA user I/O pins from the parallel peripheral mode programming, the FPGA can now allow inputs from dual channel camera configurations (ML-C3-2048K model).

Additional Algorithm Memory

With the extra FPGA resources being freed, a new sub-system in the FPGA logic is created to supply 64 bits of continuously addressable memory (flip-flops) and 256 bits of partially addressable RAM (16x16) to any video processing algorithm. With these programmable values (during FPGA operation), algorithms may become more complex by using tunable registers and look-up tables.

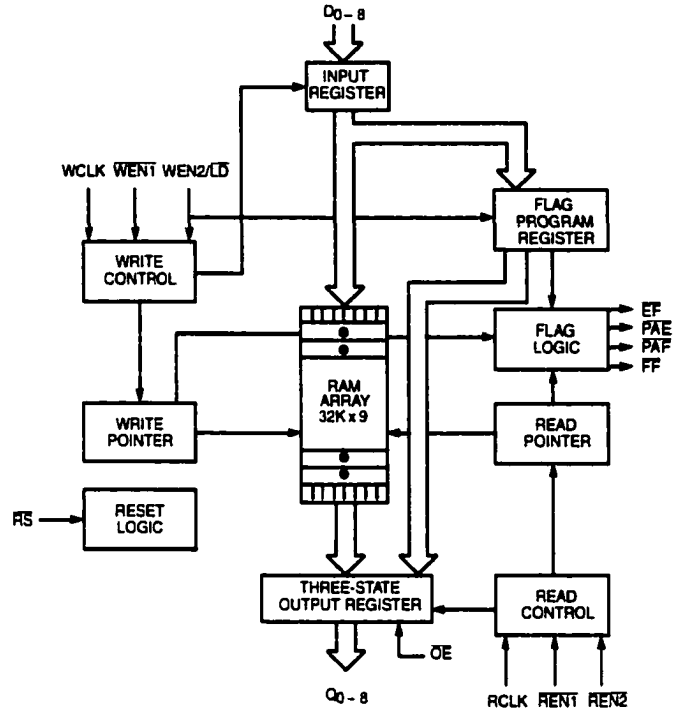
3.3.2 SRAM Simulated FIFO

The simulated FIFO in the first generation system is inadequate for this application due to the 3 cycle read limitation which creates the possibility of FIFO overflow. A missed write operation will interrupt the system's functions by corrupting both image and synchronization data. Therefore, the second generation system uses external synchronous programmable FIFO chips.

Cypress CY7C4271 Synchronous Programmable FIFO

The Cypress CY7C4271 synchronous programmable FIFO [3] is a high speed 9-bit wide, 32K byte deep memory with clocked read and write interfaces. This FIFO can perform single cycle simultaneous read and writes operations with the advantage of increasing the data width to that greater than 9 bits (in parallel configurations) and allowing for "almost" full and "almost" empty flags which are programmable based on the current FPGA algorithm. These FIFOs require only 4+w (w=data width) data lines (write host) and 4+w data lines (read host) for any FIFO depth (see Figure 3.2). This is a savings compared to the first generation system which required 2+w+d (d=$\log_2$ of depth) interfacing pins.

Figure 3.2 Block Diagram of Programmable Synchronous FIFO



Interface and Programmability

The synchronous programmable FIFOs have all their control logic imbedded into the package thus only requiring simple write control logic to handle almost all flag programming and resets from within the FPGA (23 CLBs in the second generation, 68 CLBs in the first generation). This has increased the number of CLBs available to the real-time processing algorithms. The width of the FIFO output is 32 bits (see Section 3.4.2), therefore either three or four FIFO devices are required. In the interest of minimizing potential PCB congestion, three FIFOs with a total width of 27 bits is chosen.

3.3.3 RS-232

The performance of the first generation system demonstrates that RS-232 at 9600 baud transmission is not capable of maintaining the data rate of the simple FPGA algorithms due to the artifacts in the materials under test. If the baud rate were to be increased, hardware flow control would have to be implemented and the MCU would not be able to maintain the transmission speeds with these extra considerations. Plain RS-232 is also a

host-to-host communication protocol which is not suitable in multi-host environments. It is determined to discontinue using the RS-232 as the main host communication protocol, but to keep it for communication to the DALSA MCU in the camera.

3.4 Design Additions

3.4.1 Digital Signal Processor

Although effective image compression rates were obtained with the simple algorithms implemented in the FPGA in the first generation system, these algorithms were designed for only very specific types of patterns and textures. In the future, more complex textures and patterns may have to be analyzed and processed which may require traditional DSP arithmetic and program control. Although the Xilinx 4000E series FPGA can accommodate most complex control designs, a single multiply and accumulate (MAC) block requires a considerable amount of resources. Several of these MAC blocks will certainly not fit in a single FPGA. The FPGA also has a limited amount of memory (32 + 2 bits per CLB). To allow for the possibility of future two dimensional processing algorithms, a memory of previously scanned lines will be required. The MCU in the first generation system was too slow for the expected higher speed interfacing in the second generation system. MCUs by design are not meant to be fast processors; they are simply designed to perform low bandwidth control tasks. The solution for all the above requirements is to install a Digital Signal Processor (DSP) chip in the second generation system. A DSP chip will allow for higher speeds, faster and more efficient arithmetic, larger memory ranges and program control flexibility. The Texas Instruments TMS320C5x integer based DSP is chosen for its performance, operating characteristics and physical size.

Texas Instruments TMS320C5x

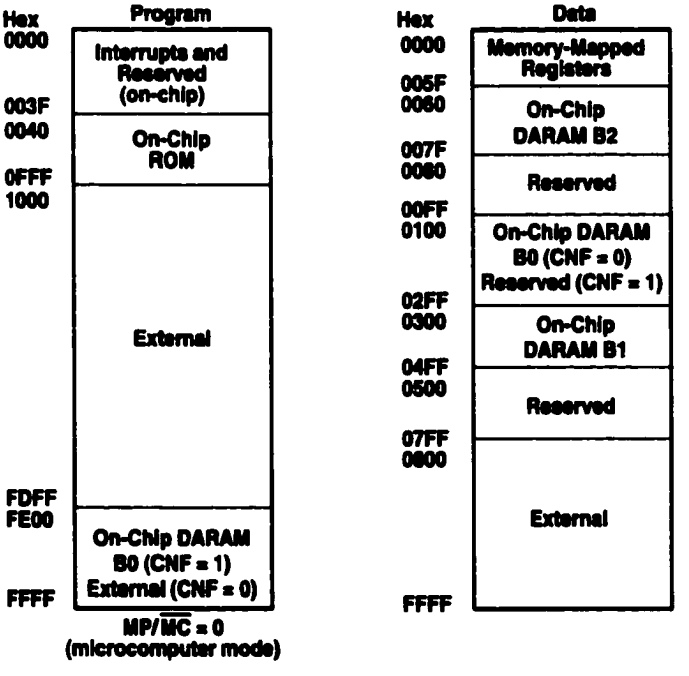
The Texas Instruments TMS320C5x DSP processor [22] is a 5V 16-bit processor with a 32 bit ALU, a single cycle 16x16 multiplier/adder, 224K words externally expandable RAM (64K words program, 64K words data local, 32K words data global, 64K words I/

O), synchronous serial ports, and operating frequencies of up to 100MHz in a single 100-132 pin surface mount chip. For this particular application, the TMS320C52B-100 DSP is chosen.

Texas Instruments TMS320C52B-100

The TMS320C52B DSP variant includes a few additional features such as 100MHz operation, a PLL clock multiplier, dual access internal memory (1056 words), external boot loader (on-chip ROM) and a smaller 100 pin surface mount package (See memory layout in Figure 3.3).

Figure 3.3 TMS320C52B Memory Map

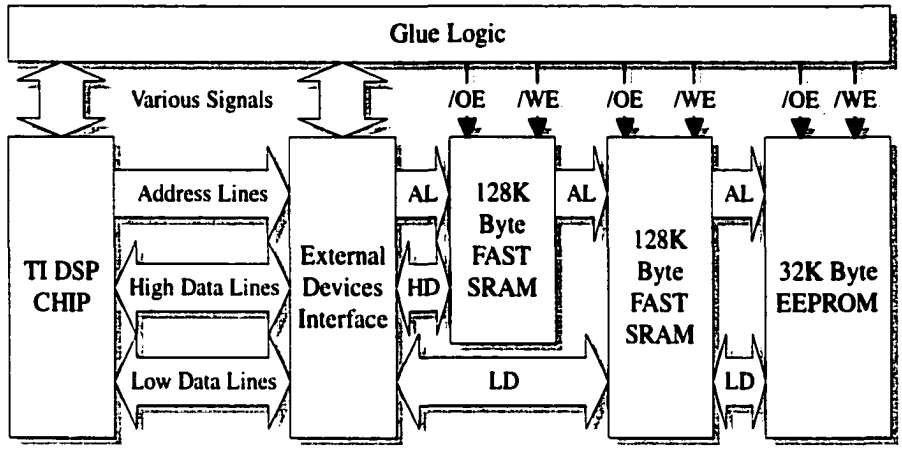


External Memory

For relatively large image processing purposes, adding external memory to the DSP bus is required. Since the processing algorithms are unknown at design time and can be quite memory intensive, the full addressable range of program and data memory is added. The I/O memory is reserved for connectivity to external devices. The approach to be taken is adding two 128K byte SRAM devices [17] in parallel (combined width is 16 bits) for the

program and local data memory plus a 32K byte EEPROM for global data memory, which is used for DSP parallel boot up mode requiring only an 8 bit data path (see Figure 3.4).

Figure 3.4 Block Diagram of DSP and Added External Memory



External Device Interfacing

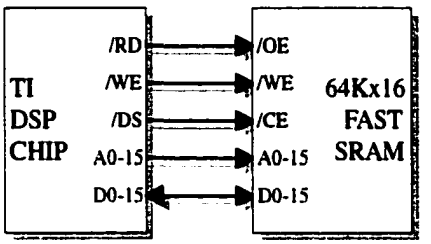
Traditionally, TI DSP chips do not offer a simple external memory interface and some external interface logic is required. This problem is addressed in the TMS320C5x series to an extent where additional signals are provided to give a more seamless interface to single memory devices; however, it is not effective for multiple memory systems. In order to connect external devices (such as RAM, or FIFOs), several signals from the DSP chip must be interpreted properly. There are two methods which can be used to interface with the DSP: asynchronous and synchronous.

Asynchronous Device Interfacing

The asynchronous method is the preferred method when connecting to memories since most memories offer asynchronous interfaces. When connecting to a single asynchronous memory, all that is required is that the speed of the memory be faster than the speed of the DSP chip. Figure 3.5 shows a DSP chip connected directly to a single memory device. In this case the DSP is only capable of addressing external local data since the data strobe ($\overline{DS}$) signal is connected to the memory's chip enable ($\overline{CE}$). The TMS320C5X's extra

memory signals ($\overline{RD}$, $\overline{WE}$) are asserted at the proper cycle times to ensure reliable read and writes without extra logic.

Figure 3.5 Single Asynchronous Memory Device Connection



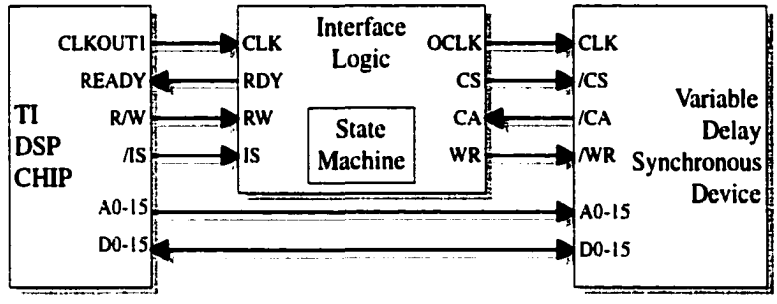
It is also possible to add external program memory in parallel by connecting the program strobe ($\overline{PS}$) to another memory's chip enable, however, in practice a 64Kx16 memory is not a very common device and is very difficult to find. This method is flawed in the case of global data where the DSP's global strobe ($\overline{BR}$ & $\overline{DS}$) would also need to be evaluated; this would require external logic since most memories do not offer many chip enable functions. In addition, most memory chips currently available, enter into a low power state when the chip enables evaluate to *false*. Typically, when the memory is enabled again, there is a time delay before they are capable of operating and this time is much greater than that of the DSP memory cycle; this results in missed read/write operations. Therefore, the external logic method is favoured, in this design, to the chip enable approach which is not practical for high-speed designs.

Synchronous Device Interfacing

The synchronous method is necessary for timing sensitive devices (such as a FIFO). To properly connect to devices in this manner, a clock reference is needed. The DSP chip offers two clock sources: $\overline{STRB}$ and CLKOUT1. The $\overline{STRB}$ line is changed only during external device access, while CLKOUT1 is derived from the actual DSP clock source. Since the interface logic for the system may be required to be operating even when the DSP chip is not addressing external devices, the CLKOUT1 signal is the best choice for a clock driver. The DSP chip also supports external waitstates (by probing the READY signal at the end of clock cycle) which are sometimes necessary in slower or delayed

external devices. During an external device access, the DSP's signals (A0-A15, D0-15 for write, $\overline{IS}$, $\overline{PS}$, $\overline{DS}$, $\overline{BR}$ and R/W) are all valid on the falling edge of CLKOUT1. By evaluating most of these signals and setting the READY line, a state machine in the external logic can guide data from an external device to the DSP's bus or vice versa (See Figure 3.6).

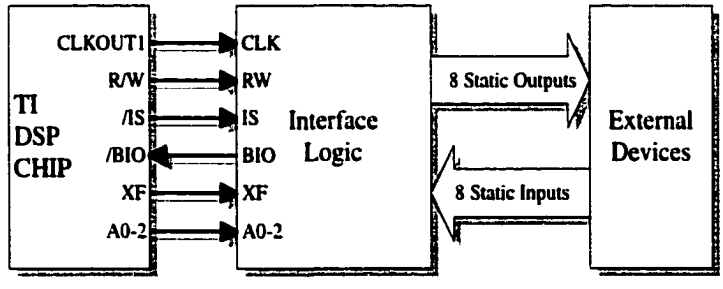
Figure 3.6 Synchronous Device to DSP Connection



Static Input/Outputs

A DSP chip, unlike the generic MCU, is not designed to have static I/O signals. TMS320C5x series DSPs only have 2 static I/O signals, one input ($\overline{BIO}$), one output (XF). In order to add more static I/O signals, additional logic will be added to the external devices interface to create a virtual device which sets $\overline{BIO}$ or evaluates XF based on an address driving these static signals (see Figure 3.7).

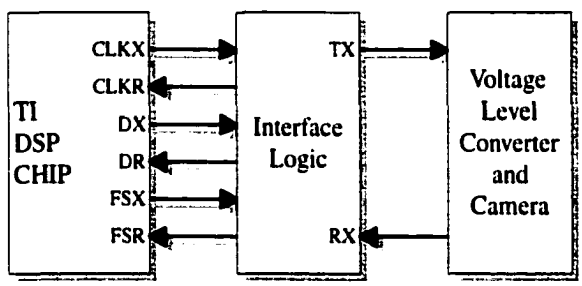
Figure 3.7 Implementation of Static I/O with DSP and Extra Logic



Asynchronous Serial Communication

In order to communicate to the DALSA MCU, a RS-232 communication protocol is required. However, the TMS320C5x series DSPs only contains synchronous transmission controllers. The signals from these controllers can be modified with the aid of extra logic to add/remove synchronization pulses & start/stop bits to act as though the signals are from an asynchronous communication system. Hardware handshaking signals required by the DALSA MCU can be implemented using the static input/output system described previously. These signals must then be converted to the appropriate RS-232 voltages with a voltage level converter. These modifications will also be implemented in the external "glue" logic (see Figure 3.8).

Figure 3.8 Synchronous to Asynchronous Transmission Converter



3.4.2 Camera To Host Communication

As mentioned in Section 3.3.3, the RS-232 communication protocol in the first generation camera is changed from the main host communication system to a secondary system, since the controller requires a significant amount of overhead for high speed transmission and is not designed for networking environments.

In order to connect numerous cameras to a single host, a networking system is needed to provide high speed transmission, low or predictable transmission latency and low cost. Several network options were considered.

Ethernet

Ethernet, being the most popular network solution at the time offered speeds up to 10Mbps, however, 10Mbps is not able to satisfy a single camera's maximum uncompressed data burst. The option of 100Mbps ethernet was available, but it was still in late development and very few vendors had usable chipsets to support it in embedded systems (non-PCI). 100Mbps ethernet also requires high speed switches for network interconnect to other nodes which can cost over $2000 for a few ports. Another disadvantage of ethernet is its unpredictable latency and lack of bandwidth management.

Asynchronous Transfer Mode (ATM)

Asynchronous Transfer Mode (ATM) offers speeds in multiples of 155Mbps and optical data links which are ideal for noisy environments. ATM also offers predictable latencies and Quality of Service (QOS) to regulate bandwidth between nodes. The main disadvantage of ATM is connectivity costs for fiber optic lines and connectors. An ATM switch is normally in excess of $3000 depending on the operating speed.

The above options did not meet the systems requirements; they were either not easily implementable or too expensive. It was decided to attempt an implementation using the (then) new IEEE standard for high-speed serial bus networks based on a research project known as "Firewire" originally conceived at Apple Computer Inc. in 1986.

IEEE 1394 - Firewire

The Apple Computer Inc. Firewire standard describes a low-cost high-speed serial bus environment primarily designed for connecting desktop computers to audio/video devices. Apple originally intended it to replace the SCSI, serial and printer type interface ports on desktop systems by consolidating them into a unified high-performance serial bus.

The 1394-1995 is an IEEE designation for a high performance serial bus [9] based on the original "Firewire" standard. IEEE added specifications for a backplane (for example,

VME, FutureBus+) physical layer as well as additional specifications to the point-to-point cable-connected virtual bus. The backplane version operates at 12.5, 25 or 50 Mbps, whereas the cable version supports data rates of 100, 200 and 400 Mbps across the cable medium supported in the current standard. Both versions are totally compatible at the link layer and above. The interface standard defines transmission method, media and protocol.

The primary application of the cable version is the integration of I/O connectivity to a personal computers using a low-cost, scalable and high-speed serial interface. The 1394 standard also provides new services such as real-time I/O and live connect/disconnect capability for external devices including disk drives, printers and hand-held peripherals such as scanners and cameras.

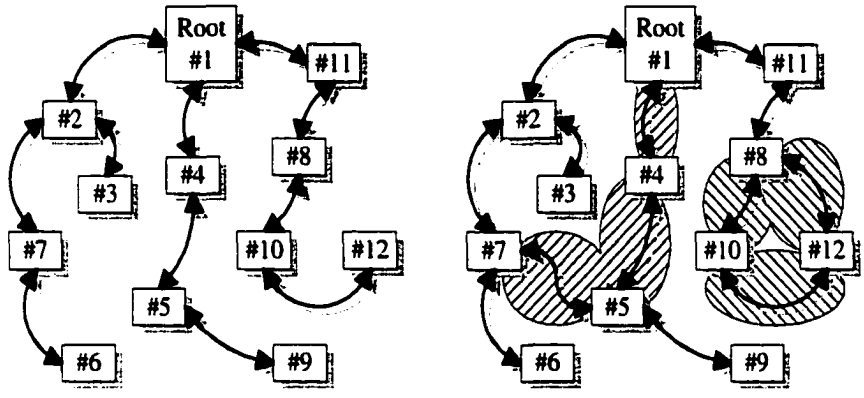
The 1394 standard is a transaction-based packet system organized as though it were memory space interconnected between devices, with the devices residing in slots on the main backplane. Device addressing is 64 bits wide, partitioned as 10 bits for network IDs (one ID reserved for local network), 6 bits for node IDs (one ID reserved for broadcast) and 48 bits for memory addresses. This results in the capability to address 1023 networks of 63 nodes, each with 281 terabytes of memory. Memory-based addressing, rather than channel addressing, views resources as registers or memory that can be accessed with processor-to-memory transactions.

Some key features of the 1394 topology are multi-master capabilities, live connect/ disconnect (hot plugging) capability, venderless cabling connectors on interconnect cabling and dynamic node address allocation as nodes are added to the serial chain. Another feature is that transmission speed is scalable from approximately 100 Mbps to 400 Mbps.

Each node may also acts as a repeater (up to 27 ports), allowing nodes to be chained together to form a tree topology. Cable distance between each node is limited primarily by signal attenuation. An inexpensive cable with 28-gauge signal pairs can be up to 4.5 meters long. Due to the high speed of 1394, the distance between each node or hop should

not exceed 4.5m and the maximum number of hops in a chain is 16, for a total maximum end-to-end distance of 72m. The only restriction on the cable topologies is that a maximum of 63 nodes can be connected in a simple no-loops tree with 16 or fewer hops (see Figure 3.9).

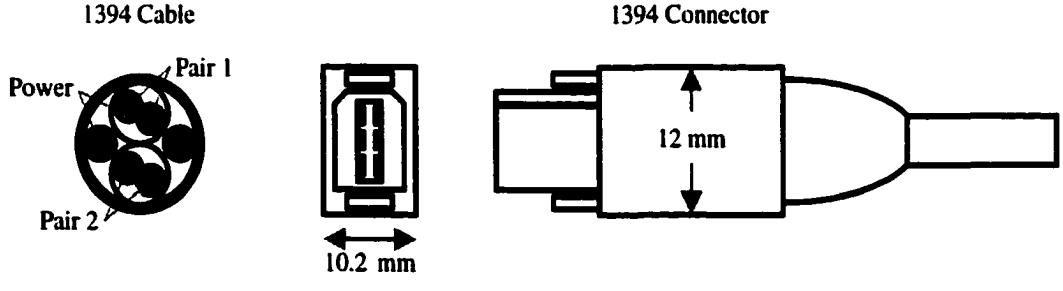
Figure 3.9 Correct and Incorrect 1394 Cable Topologies



No Loops Loops - Topology Violations

The cable environment uses a crossed three-pair shielded cable and a miniature connector to carry transmit/receive data as well as to source or sink power (between 8 and 40 VDC at no more than 1.5 A, see Figure 3.10). A unique feature of the 1394 cable version is the distribution of power through the cable for operation of the transceiver's repeating functions even if the node power is off.

Figure 3.10 1394 Cable and Connector



Both asynchronous and isochronous data transfers are supported. The asynchronous format transfers data and transaction layer information to an explicit address. The

isochronous format broadcasts data based on channel numbers (up to 32) rather than specific addressing. Isochronous packets are issued, on the average, each 125μ seconds in support of time-sensitive applications. Providing both asynchronous and isochronous formats on the same interface allows non-real-time critical applications such as printers & scanners and real-time critical applications such as video and audio to operate on the same bus.

The tree topology is resolved during a sequence of events triggered each time a new node is added or removed from the network. This sequence starts with a bus reset phase, where all previous information about a topology is cleared. The tree ID sequence determines the actual tree structure. During the tree ID process, each node is assigned an address and a root node is dynamically assigned (it is possible to force a particular node to become the root). After the tree is formed, a self-ID phase allows each node on the network to identify itself to all other nodes. After all of the information has been gathered on each node, the bus goes into an idle state waiting for the beginning of the standard arbitration process.

An additional feature is the ability of transactions at different speeds to occur on a single device medium (for example, some devices can communicate at 100 Mbps while others communicate at 200 Mbps and 400 Mbps). Use of multi-speed transactions on a single 1394 serial bus requires consideration of each node's maximum capabilities when laying out the connections to ensure that the path between two higher-speed nodes is not blocked by a device with lower-rate abilities.

To meet Firewire's original speed requirements, it was designed to operate over fibre optic cables (with copper for power). However, further research, additional involvement from other organizations, and the industry move to implementing analog circuits in digital CMOS fabrication technologies, has allowed the IEEE 1394 standard to operate on its current shielded twisted pair copper line at a much lower cost. A typical 1394 chipset costs approximately $40.00 US and does not require additional switching hardware for interconnection to other devices.

In order to implement 1394 into an embedded system, two components are required: a link layer controller (LLC) and a physical interface (PHY) (although some vendors have combined the two). The LLC provides the interface between a CPU or other bus to create properly formatted 1394 packets. These packets are then transmitted digitally to a 1394 PHY, converting this digital information into the properly timed analog signals which connect to the actual 1394 bus. For our new system, it was decided to use both Texas Instruments TSB12C01A 1394 LLC and TSB21LV03 1394 3-port PHY. Texas Instruments was one of the first vendors of 1394 to provide a wide variety of products for all applications. Although other vendors supplied 1394 products, most were designed for a PCI bus related environment (such as a PC).

Texas Instruments TSB12C01A

The Texas Instruments TSB12C01A is an IEEE-1394 standard high-speed serial-bus LLC [27] [28] that allows for easy integration into an I/O subsystem. The TSB12C01A is a 5V, 100 pin surface mount device which transmits and receives correctly formatted 1394 packets and generates and inspects the 32-bit cyclic redundancy check (CRC). The TSB12C01A is capable of being a cycle master (root) and supports reception of isochronous data on two channels. It interfaces directly to most 1394 physical layer chips and can support bus speeds of 100, 200, and 400 Mbps. The TSB12C01A has a generic 32-bit host bus interface which makes connection to most 32-bit host buses very simple and is interrupt driven to reduce host polling. The TSB12C01A has software-adjustable FIFOs for optimal FIFO size and performance characterization and allows for variable-size asynchronous-transmit FIFO (ATF), isochronous-transmit FIFO (ITF), and general-receive FIFO (GRF).

TSB12C01A Memory Map

All of the TSB12C01A registers and FIFOs are addressable through direct memory addressing as 32-bit values is shown in Table 3.1.

Table 3.1 TSB12C01A Memory Map

Memory Range	Explanation
0x00-0x07	Information Registers (Version, Node ID)
0x08-0x0B	Control Registers
0x0C-0x13	Interrupt and Interrupt Mask Registers
0x14-0x1B	Isochronous Communication Registers
0x20-0x23	Diagnostic Registers
0x24-0x27	PHY Communication Registers
0x30-0x3F	FIFO Sizing/Status/Resetting Registers
0x80-0x8F	Asynchronous Transmit FIFO (ATF)
0x90-0x9F	Isochronous Transmit FIFO (ITF)
0xC0-0xC3	General Receive FIFO (GRF)

TSB12C01A Interfacing

The TSB12C01A 32-bit interface operates as a simple clocked hardware handshaking system typically used in a MCU (similar to Figure 3.6). The TSB12C01A is first generation 1394 TI product and it is based on the original Apple Firewire cores under license. Because of this, these devices are not optimal and are somewhat problematic (under excessive asynchronous transactions), but upgradable to the new devices (to be released in mid 1999) which support true synchronous communication (one cycle read and writes). According to the documentation a read/write operation can take up to 9 clock cycles, which translates to a minimum of 15MBps burst across the 32-bit bus. In practice, however, the TSB12C01A is found to only transmit up to 4MBps maximum.

Texas Instruments TSB21LV03A

The Texas Instruments TSB21LV03A is a 3.3V device which provides the analog physical layer functions needed to implement a three-port node in a cable-based IEEE 1394-1995 network [30] [31]. Each cable port incorporates two differential line transceivers. The transceivers include circuitry to monitor the line conditions as needed for determining connection status, for initialization and arbitration, and for packet reception and transmission up to 200Mbps. The TSB21LV03A is also inter-operable with 5V 1394 LLC and PHYs using separate 5V supplies.

Galvanic Isolation

The IEEE 1394 cable is designed to transmit and receive data at various data rates and also to source and/or sink power to/from remote nodes. This allows remote nodes that either do not have their own source of power or have their power turned off to continue to function in the 1394 network. Since multiple nodes are allowed to source power simultaneously, this can cause grounding problems in the system if galvanic isolation between nodes is not handled properly.

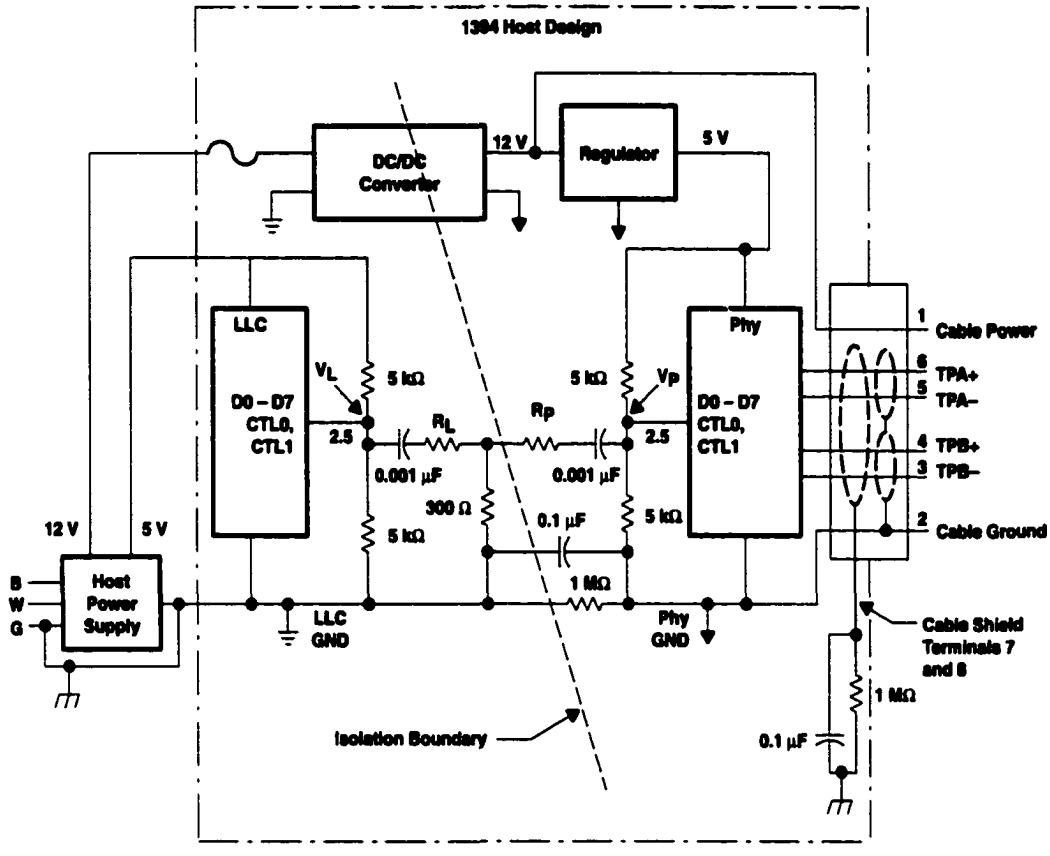
Due to the distances allowed between nodes in a 1394 network, different nodes on the network can be plugged into AC outlets that are in different ground domains, commonly referred to as green-wire grounds. The potential difference between these grounds can be DC, AC at 60 Hz along with its harmonics, and various noise components. If these grounds are connected together by the 1394 cable logic ground or shielding, a ground loop exists and current flows into the cable. These ground-loop currents can have several negative effects on the 1394 network, which includes degradation of data signals on the cable, excessive EMI from the cable, ground currents high enough to damage components in the system, and if the potential difference is large enough, a personal shock hazard.

Annex A in the IEEE 1394-1995 standard discusses the various places that galvanic isolation can be implemented when necessary to achieve galvanic isolation of the node.

One of the most cost effective places to isolate a node is at the PHY-LLC interface due to the relatively small number of signals that need to be isolated. Annex J of the IEEE 1394-1995 standard illustrates two techniques to achieve PHY-LLC galvanic isolation. One method uses capacitive-isolation circuitry and the other uses a transformer-isolation circuit on each PHY-LLC signal. The capacitive isolation-circuit is able to galvanically isolate up to the working voltage of the capacitors used in the circuit. The transformer circuit would typically be able to galvanically isolate to higher voltage levels due to its generally higher voltage capability between primary and secondary windings.

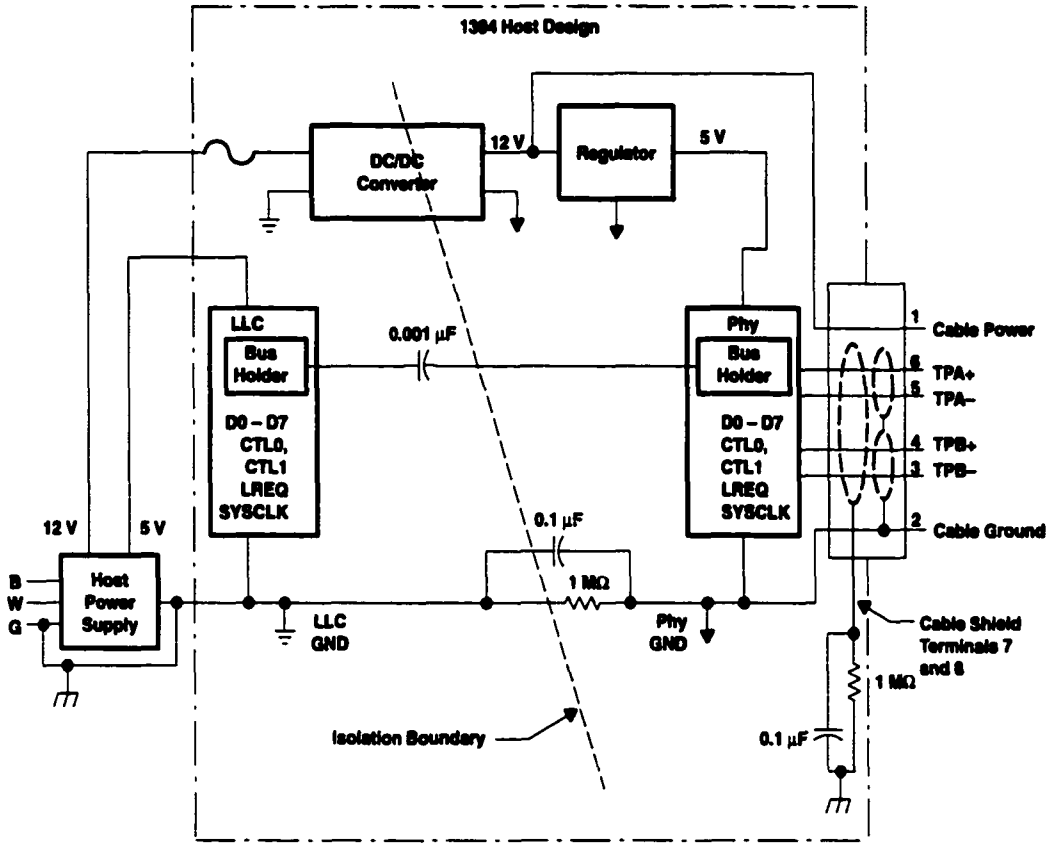
Figure 3.11 shows a capacitively coupled PHY and LLC that uses the capacitive-isolation circuit described in Annex J of the standard. Each bidirectional signal requires the network shown in the figure. This method requires that both the PHY and LLC support the galvanic isolation method in Annex J (for the TSB21LV03A & TSB12C01A, the $\overline{\text{ISO}}$ line must be pulled low).

Figure 3.11 Annex J Method of Galvanic Isolation



Texas Instruments provides a simplified technique that uses bus-holder circuits [19] which are required on the input side (both sides when bidirectional) of a single capacitor that will form a galvanic isolation barrier (see Figure 3.12). This method does not require the PHY and LLC to support the Annex J requirements, instead a simple CMOS signal connections between the PHY and the LLC suffices.

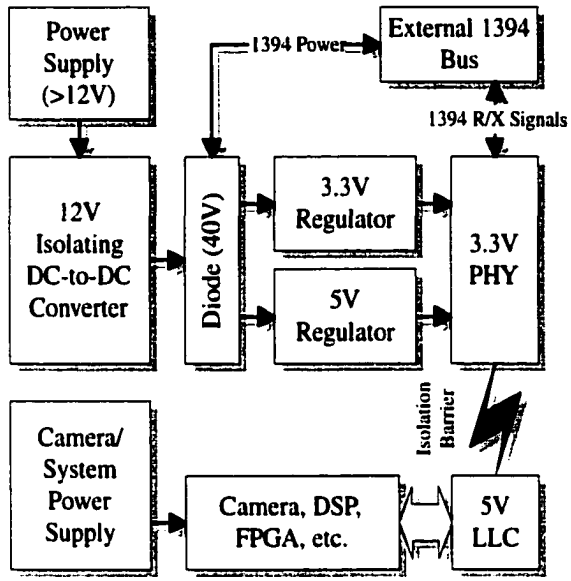
Figure 3.12 TI Method of Galvanic Isolation



For a 200Mbps maximum system, the Texas Instruments method of galvanic isolation requires only 8 capacitors, whereas the method described in Annex J of the standard requires 14 capacitors and 48 resistors. As well as having less components, the Texas Instruments method also performs better in other areas as well [30].

Since the design of the in-camera processing boards is very area limited, the TI method is the one used in implementing galvanic isolation with the PHY and LLC connections. In fact, the TSB21LV03A supports this method internally and does not require any external bus holder circuitry. However, since the PHY is a 3.3V device and the LLC is a 5V device, two regulators providing 3.3V [11] and 5V [12] from the cable or camera power supply are needed to power the PHY and it's LLC interface (see Figure 3.13).

Figure 3.13 3.3V PHY Interface to 5V LLC



3.4.3 Programmable Logic

In order to have all the above components integrated into a complete working system, a "glue" logic device is necessary that will store the programmable logic semi-permanently. The device needs to be easily reprogrammed several times, either while being in the system, or when removed from a socket connection. There are several devices on the market which could accomplish this task, but it is necessary to find a device which will also integrate into the already existing tool set thus avoiding the need to invest time and money into another series of design tools. The choice is to proceed with a Xilinx 9500 Series CPLD device since it is easily integrate into the existing FPGA design flow with barely no changes and no additional costs.

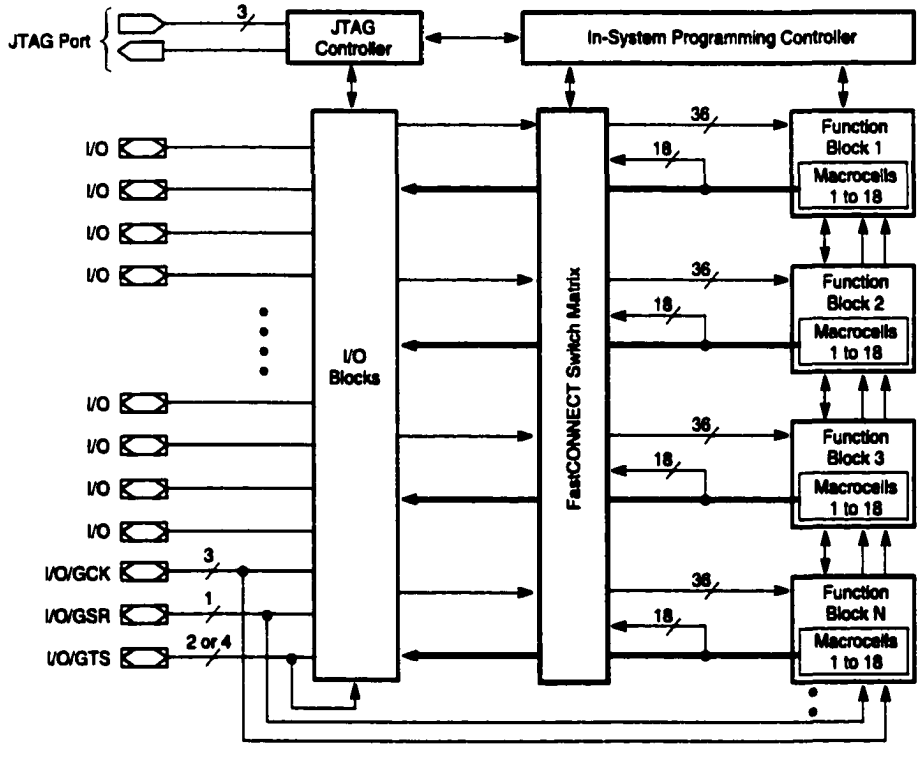
Xilinx 9500 Series CPLD

The XC9500 CPLD family [39] provides in-system programming capabilities for high performance and general purpose logic integration. All devices are in-system programmable for a minimum of 10,000 program/erase cycles using Xilinx's FastFLASH memory. The XC9500 architectural features address the requirements of in-system programmability (JTAG: IEEE 1149.1 boundary scan). Enhanced pin-locking capability

avoids PCB reworking. In-system programming throughout the full device operating range provide worry-free reconfigurations and system field upgrades. User I/Os may be configured for 3.3V or 5V operation and outputs provide 24 mA drive.

Each XC9500 device is a subsystem consisting of multiple Function Blocks (FBs) and I/O Blocks (IOBs) fully interconnected by the Xilinx's FastCONNECT switch matrix (see Figure 3.14). The IOB provides buffering for device inputs and outputs. Each FB provides programmable logic capability with 36 inputs and 18 outputs. The FastCONNECT switch matrix connects all FB output and input signals to the FB inputs. For each FB, 12 to 18 outputs (depending on package pin-count) and associated output enable signals drive directly to the IOBs.

Figure 3.14 Xilinx 9500 Series CPLD Internal Structure



Preliminary Logic Design

A preliminary logic interface to connect all the system components is designed in order to estimate the number of macrocells so an appropriate CPLD device can be selected. The

Xilinx CPLDs are available in the sizes of 36, 72, 108, 144, 216 and 244 macrocells. Unfortunately, as the number of macrocells increase, the delay of the device also increases. Due to the minimum cycle speed of the DSP chip (requirements for the READY line), the minimum speed CPLD is selected; a 9536 in order to meet the zero waitstate memory/device access.

Xilinx 9536 CPLD

The 9536 CPLD [40] is available in a 44 pin surface mount package with 34 user I/O pins available and a maximum pin-to-pin delay of 5ns. Although the 9536 meets the speed specifications, it does not meet the I/O and macrocell specifications. The preliminary design is very resource intensive since it must convert the DSP's 16-bit bus to and from the 32-bit 1394 LLC and FIFO buses. This design is also quite elaborate since it uses both read and write caches to transfer data to the 1394 LLC and from the FIFOs to eliminate additional DSP waitstates. This design requires a 95288 device and therefore is not efficient for this system, hence a compromise is needed.

3.4.4 Bus Exchanger

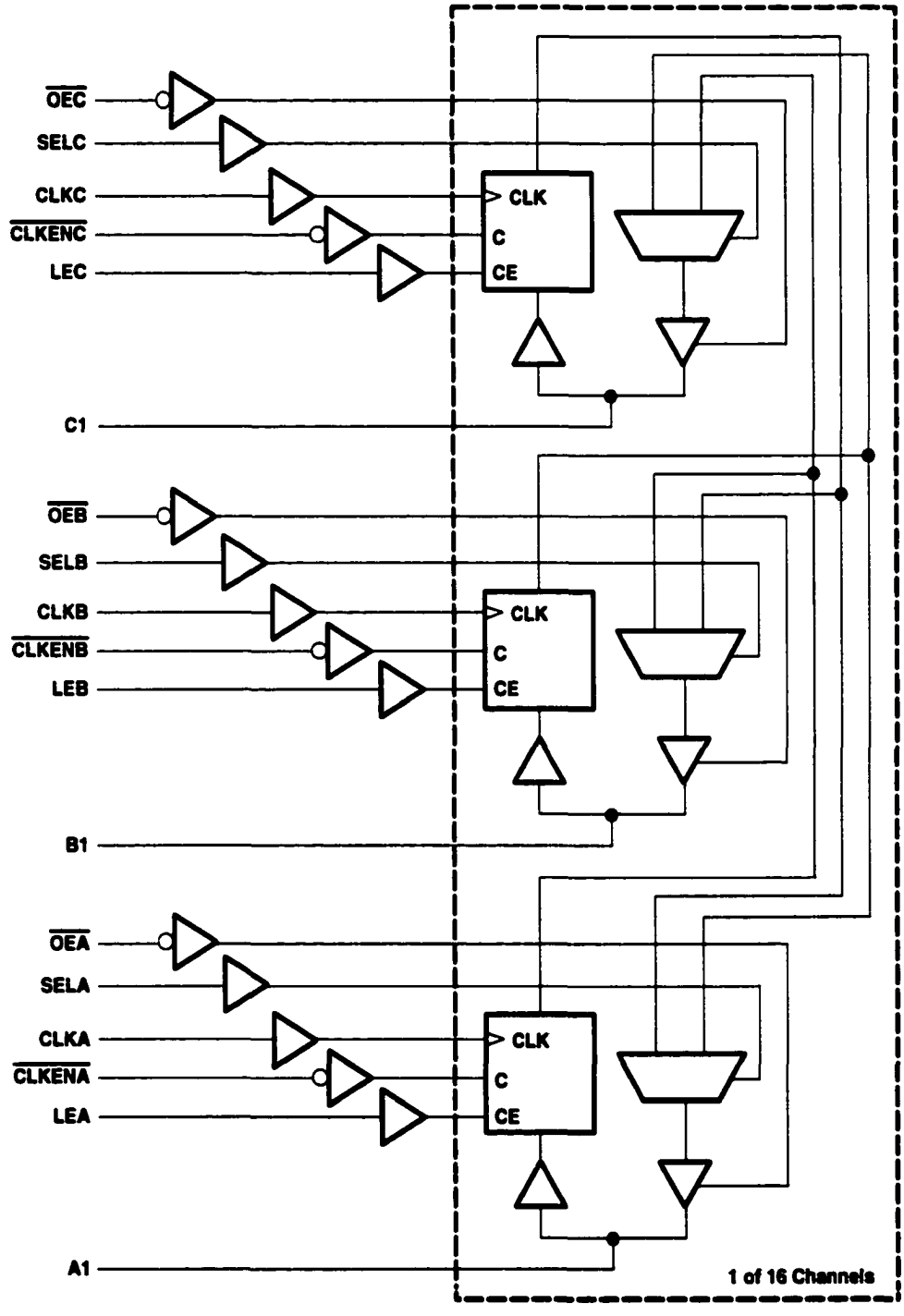
In order to reduce the number of macrocells for the interface logic, a bus exchange device is added. The device chosen is a Texas Instruments SN74ABTH32316 universal bus exchanger.

Texas Instruments SN74ABTH32316

The Texas Instruments SN74ABTH32316 consist of three 16-bit registered I/O ports [18]. These registers combine D-type latches and flip-flops to allow data flow in transparent, latch, and clock modes. Data from one input port can be exchanged to one or more of the other ports (see Figure 3.15).

By statically and dynamically setting several signals of the bus exchanger, two 16-bit writes from the DSP chip can be converted to a 32-bit value or a 32-bit value can be converted to two 16-bit DSP reads (see Appendix D.3.2 "VHDL Code" on page 438).

Figure 3.15 Bus Exchanger Internal Structure



By using the SN74ABTH32316, the requirements for the interface logic are greatly reduce; however, the LLC and FIFO buses are unified and the performance enhancing caches are removed. This new interface requires more than 36 but less than 72 macrocells. Therefore, two 9536 CPLDs are used to implement the external device interface logic.

3.5 Envisioned System

3.5.1 Component level

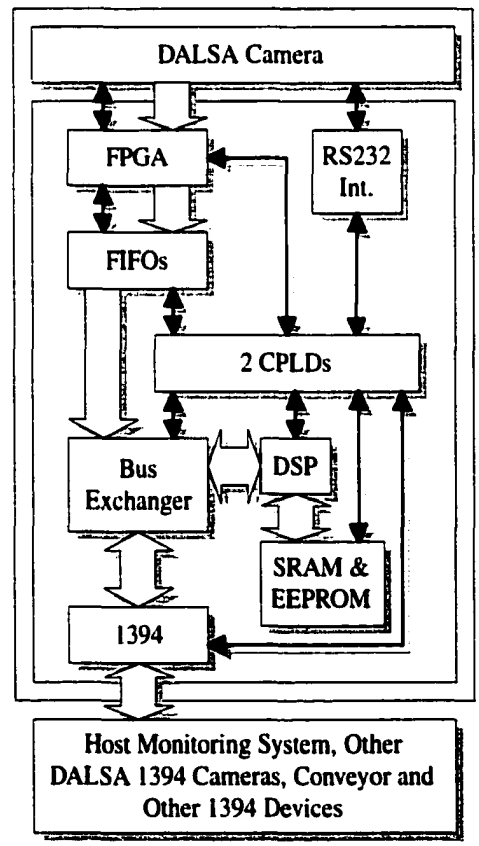
After examining the first generation system and the new component data sheets, the envisioned component level in-camera system design is shown in Figure 3.16. The DSP chip (as with the MCU in the first generation system) is the main system controller. The DSP chip has limited direct and high bandwidth interfaces to almost all devices in the system. These limited direct connections (through the interface logic) exist to maintain system control with reset lines and serial communication. The high bandwidth interface is necessary for processing the large amounts of video data during regular system operation. After reset, the DSP chip is the only active device which boots up its code from the external EEPROM device (DSP built-in parallel boot-up mode). The firmware code will reset and place all on board devices in IDLE mode until a command is received from the host, which is connected to the system through the 1394 interface. The firmware is therefore required to have a complete 1394 network stack (interrupt driven); however, other DSP capabilities such as FPGA programming and camera data processing algorithms are not necessary to be present in the firmware since they can be downloaded from the host when needed.

Ideally, after system reset, the DSP receives the following commands from the host to perform the video processing/capture operation:

1. Program the DALSA MCU (for camera settings)
2. Download the FPGA bitstream (hardware algorithm)
3. Download DSP algorithm (software algorithm)

4. Program the FPGA
5. Set the FPGA algorithm parameters
6. Reset the FIFO
7. Begin hardware processing (start the video capture)
8. Begin processing the incoming FPGA stream (software processing)
9. Transmit the processed data to the host monitoring system
10. Stop processing incoming FPGA stream
11. Stop hardware processing

Figure 3.16 Envisioned Component Level Block Diagram

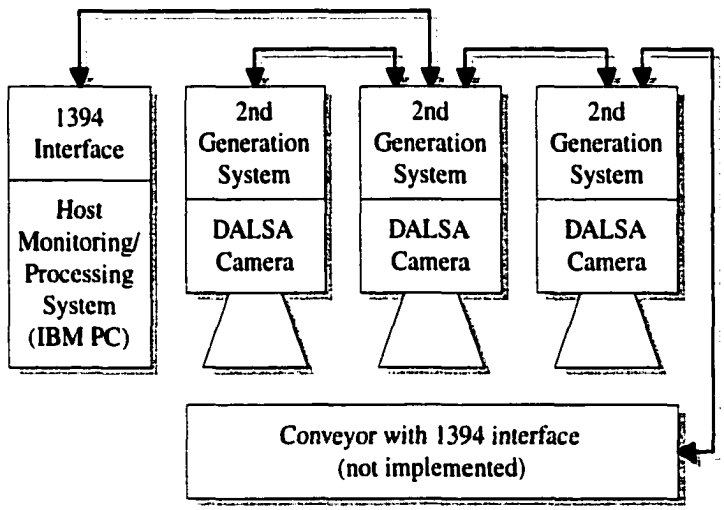


3.5.2 System Level

The high-level system block diagram is shown in Figure 3.17. The host monitor/ processing system (an IBM compatible PC) with a 1394 interface card is connected to one

or more DALSA cameras with the second generation processing boards installed. Although not implemented for this project (perhaps in a third generation system), a more sophisticated conveyor could also connect to the cameras and the host supplying the synchronization signal across the network.

Figure 3.17 Envisioned System Level Block Diagram



3.6 Summary

By thoroughly examining the first generation system and the objectives of this work, a selection of *available* components is made from a variety of vendors to meet the requirements of the envisioned system. This system consists of:

- a Xilinx FPGA and two fast CPLDs
- a TI DSP chip, Bus Exchanger and 1394 LLC, PHY and busholders
- Memory devices from Samsung and Xicor
- Isolating DC-to-DC converter and voltage regulators
- Power, 1394 and DALSA camera connectors
- Many capacitors and resistors for power control, logic control and 1394 interfacing

These components and devices will be integrated together to create an interfacing system that embeds real-time video processing and network connectivity within a single camera.

Chapter 4

Second Generation System Implementation

4.1 Introduction

Upon making the final decisions for the components and their connectivity in the second generation system, a full design schematic and PCB layout is implemented with consideration to maintaining the physical design constraints.

4.2 Design Implementation

The complete second generation schematics and PCB layouts can be found in Appendix C.1 "Schematics" on page 300 and Appendix C.2 "PCB Layouts" on page 315 respectively. The same series of tools used in designing the first generation system are also used in the design of this system.

4.2.1 Physical Constraints

The Printed Circuit Board (PCB) area inside the camera is limited to an area defined by the camera casing which is 3.25" by 3.25". If any system does not fit within this physical area, it will need to be partitioned into smaller designs to fit on to multiple PCBs. In order

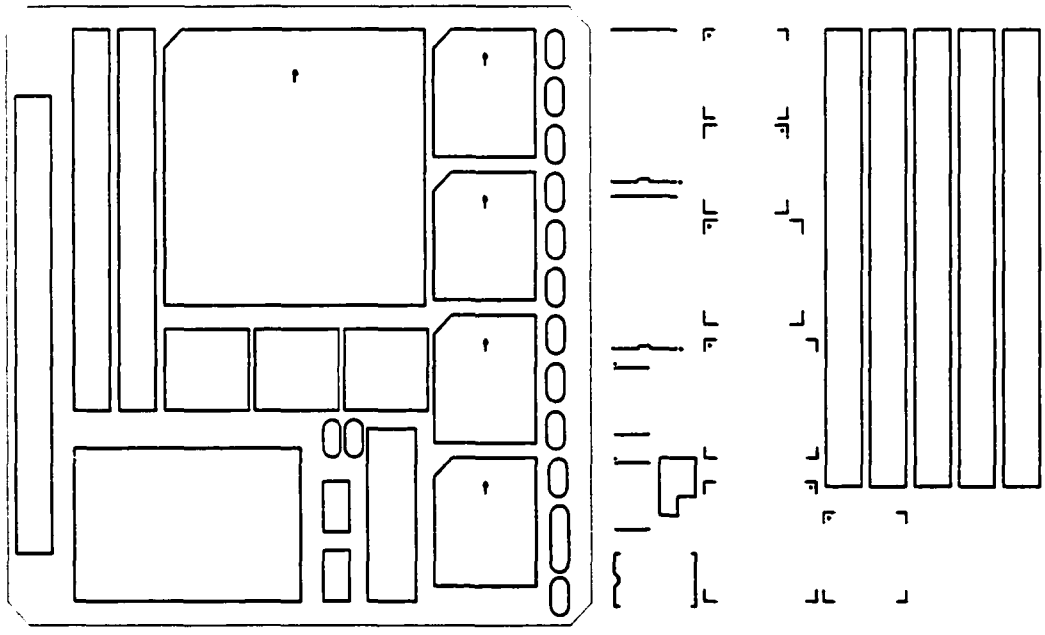
to connect these PCBs, vertical connectors (40 connections) are required which will each consume 0.25" section of board area.

The PCBs are manufactured as 4 layer boards, with a top and bottom layer and two intermediate layers for power and ground. All active components such as the FPGA, DSP and other packaged devices are placed on the top layer, while passive devices such as resistors and capacitors are placed on the bottom layer. This method is ideal for debugging since any signal trace can be removed or added (30 gauge wire) after PCB manufacturing and signal routing is simplified since power and ground routing is trivial.

4.2.2 Partitioning

A single PCB (see Figure 4.1) is insufficient to accommodate the increased number of components, devices and routing within the second generation system.

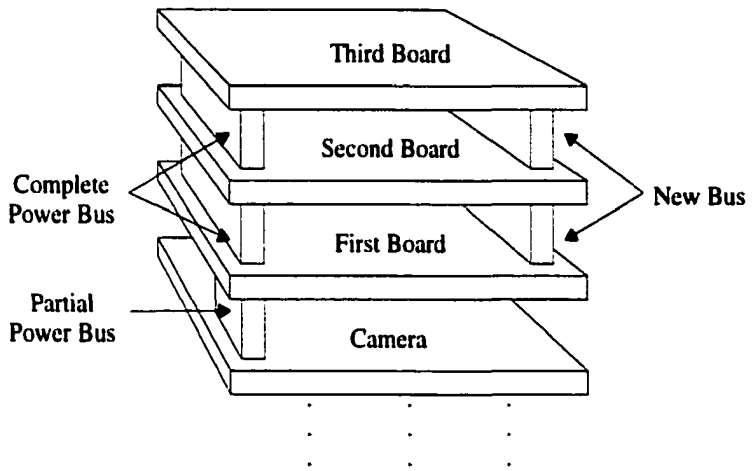
Figure 4.1 Actual Board Space and All Component Foot Prints



The stock interface board for the camera contains connections for 3 separate buses: the power bus, the digital video bus and the option bus. All of these buses are required to properly interface with the camera and the second generation system. Only the first 16

signals of the power bus are connected to the second generation system, while the remaining 24 signals contain only internal camera information and are reused in the second generation system. All the boards are connect to a portion of the power bus and the first board contains connections to the other two buses. The remaining two boards have a new bus consolidating all the necessary signals from camera and the first board (see Figure 4.2).

Figure 4.2 Power Bus Connectivity



First Board - FPGA System

The FPGA socket and RS-232 voltage converter are placed on the first board since they have the most connections to the camera system. The remaining space on the board allows for the FIFOs to also be placed which are connected to the new vertical bus containing the shared 32-bit bus.

Second Board - DSP System

The 1394 system is restricted to the third board because the component form factors [7] make board stacking difficult. The remaining DSP system is placed on the second board. The DSP system requires many internal signals which are not needed on interconnecting vertical buses. The DSP chip and the extra SRAM memory are placed along with the EEPROM socket (EEPROM must be removed for programming and testing). The DSP

system also includes the bus exchanger to connect on to the shared 32-bit bus. Since the DSP chip is the main controller, it is necessary to have the CPLDs on this board as they support the static I/O (primarily connected to the FPGA and camera's RS232) and bus exchanger controlling logic. This system also requires an oscillator to drive the DSP's clock which is placed on the solder side of the PCB.

Third Board

The third and final board contains all of the components for the 1394 system. Since this system requires galvanic isolation, meticulous planning is necessary to ensure all components are located on the appropriate ground and power planes. The LLC is placed directly beside the shared 32-bit bus along with the control signals from the CPLDs on the DSP board. A small gap is reserved for the isolating capacitors and a bus holder for the LLC is placed to implement the TI method of galvanic isolation. The 1394 connectors are mounted vertically to save PCB area and the PHY is placed according to the area demanded by the analog components for the physical 1394 connection. The DC-to-DC converter is placed onto the PCB corner along with the 3.3V and 5V regulators all of which are on separate power/ground planes. Since this is the final board, the power connector for the camera is also placed here.

4.2.3 Manual Routing

The task of routing is performed manually since automatic routing was unsuccessful at meeting the limited 4 layer constraints of the PCBs. Auto-routers generally require many board layers with the intent that implementation time will be fast and inexpensive, but at a higher manufacturing cost for more layers. The manual routing of the three PCB required approximately 4 weeks where some design changes were made to improve signal routing.

Power Routing

Since the internal two layers of each PCB contained power and ground, routing is very simple by placing a through-hole and connecting the trace to the appropriate plane. Each

power/ground pair on all active devices has a 0.001µF capacitor placed to prevent power loss to the device in case of excessive current spikes. Each board also has two to four 10µF capacitors placed to smooth out the power.

Signal Routing

After power and ground routing is performed, signal routing begins. Each component's final placement depends on the signal routing in order to obtain the most area efficient implementation. Signal widths are 0.5mm at angles of 0, 45 and 90 degrees.

4.2.4 Verification

Due to the complexity of the second generation system, a verification tool is used to determine if the PCB layout violates any design rules or deviates from the schematic netlist.

4.3 PCB Tests

4.3.1 Post Fabrication Testing

For the first test of the PCB, all traces, power and ground planes are tested for shorts to confirm that the physical design is free of manufacturing errors prior to any soldering. See Appendix C.3 "Fabricated Boards" on page 318.

4.3.2 Pre Power Up Testing

Once all components and devices are soldered on the PCBs, another test for shorts on the adjacent traces, power and ground places is performed. Any problems have to be resolved before power up. See Appendix C.4 "Assembled Boards" on page 321.

4.4 Software Tools

4.4.1 DSP Assembler/C Compiler

A TMS320C5x programming suite including an assembler and C compiler is obtained to aid in the programming of the DSP device. However, this particular architecture does not implement C code and functions well since the hardware stack is only 8 words long. To resolve this a special software stack is implemented using pointer manipulations; however, the performance is not optimal [21]. With the addition of special hardware on the system, access to these devices requires that specific operating conditions are met. To be certain that all interfacing operations are performed correctly, all code is written in assembly language to obtain the most efficient usage of device interfacing and memory [20].

4.4.2 Microsoft Visual Studio v6 and Adaptec 1394 API

The target operating system, Windows NT 4.0, architecture does not allow for direct implementation of the IEEE 1394 interface; however, Windows 98 and Windows 2000 are somewhat capable. An Adaptec (third party) card and software API is used to interface to 1394 devices within the Windows NT environment [2]. This API is accessed through C library functions directly coded with the monitoring software using Microsoft Visual Studio v6. Some portions of the software code written for the first generation system are reused for this system. This software, as in the first generation system, requires special coding techniques such as multi-threading to properly communicate with 1394 devices while still maintaining Windows compatibility.

4.5 Individual System Test

Each board is powered up independent of the camera system and other boards to verify power consumption limits.

4.5.1 Stand-Alone Tests

DSP Board

As the DSP is the main controller, tests on this system are performed first. With the aid of an EEPROM emulator, the bootup procedure is tested along with patterned RAM tests and CPLD interface tests (static I/O, FPGA and 1394 signals). Minor changes to the CPLD interface logic are applied here.

DSP and FPGA Boards

Tests are performed to check the DSPs ability to properly program the FPGA under serial peripheral mode along with FIFO tests to verify proper data transmission to the DSP. RS-232 communication tests (to and from an external terminal) are also performed requiring many changes to the CPLD logic to correct data receiving problems.

1394 Board

Tests are performed to check that power is being sourced from the 1394 bus to operate the PHY in repeater mode with no external camera power. Additional tests are performed to verify proper operation of the bus holder circuits for galvanic isolation.

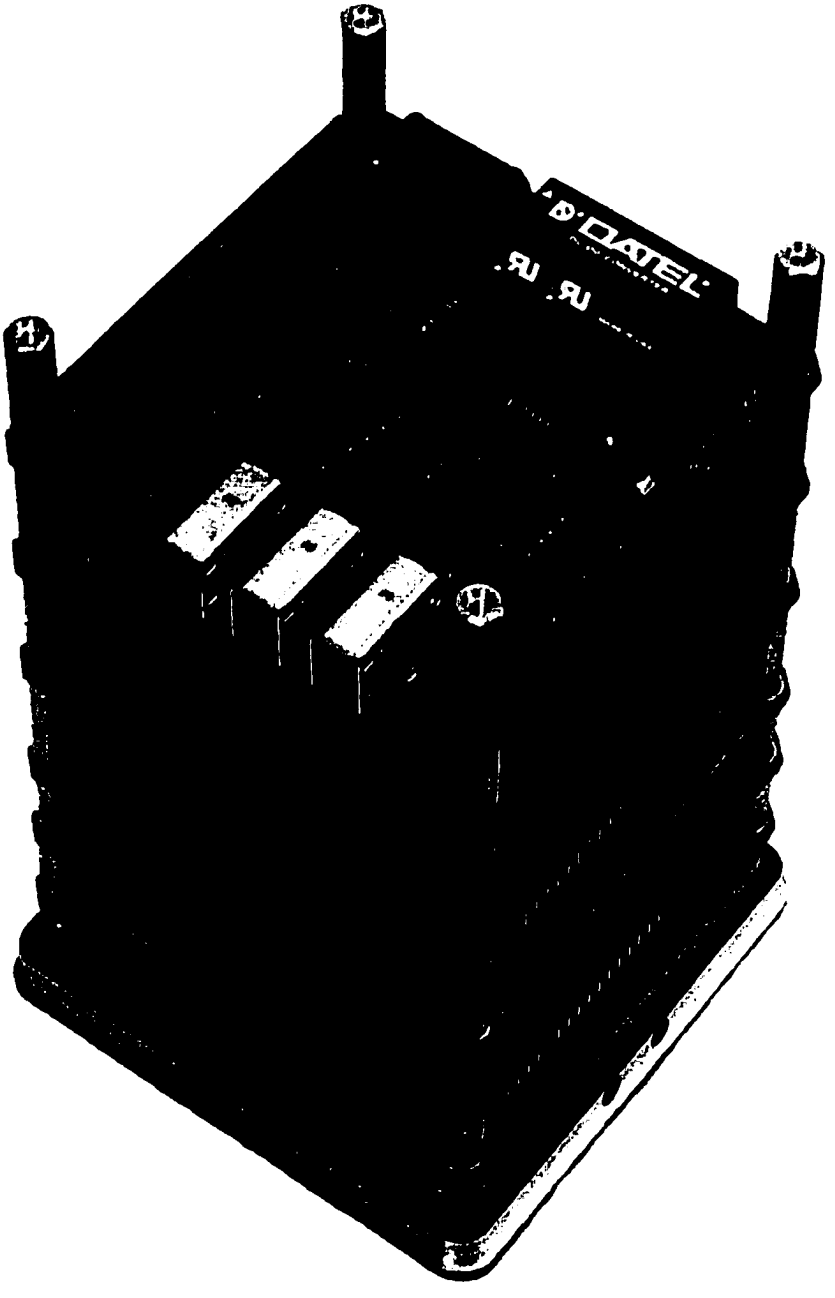
DSP and 1394 Boards

All 1394 transaction commands and bandwidth limits are tested with preliminary DSP and PC code. A watchdog timer is added to the CPLD logic to avoid DSP and system lockups during 1394 LLC communication.

4.5.2 Camera Tests

After non-camera individual board tests are performed, all boards are combined in the final system and in the camera where signal connectivity is verified (see Figure 4.3).

Figure 4.3 Second Generation System Combined with DALSA Camera



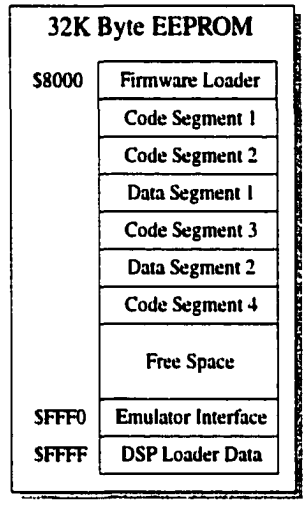
Final code writing begins by building up from the preliminary code developed for testing the individual systems. A Simple FPGA algorithm is designed to transmit unprocessed/ uncompressed data to the DSP to transmit to the host PC via asynchronous transactions.

4.6 Software Design

4.6.1 Second Generation System

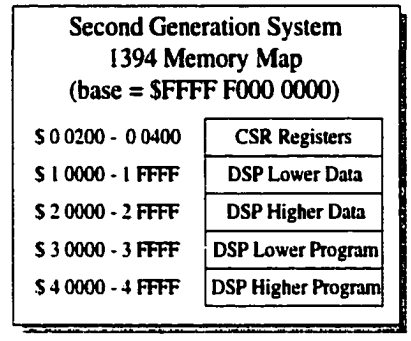
The DSP's ROM boot loader loads a small portion of code from the EEPROM which when executed intelligently loads the full firmware from the EEPROM (see Figure 4.4). This firmware may be fragmented depending on the rules provided to the object linker (See Appendix D.1.2.2 "linkcode.cmd" on page 330).

Figure 4.4 Firmware EEPROM Layout



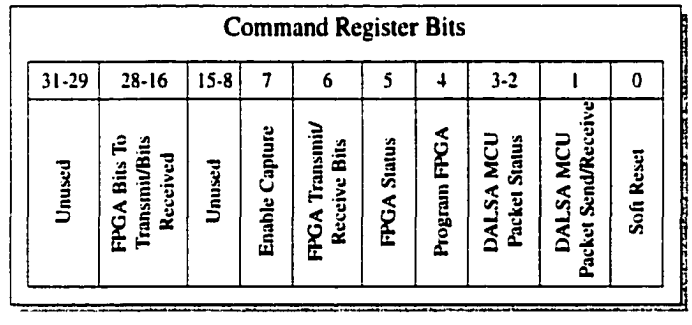
The firmware code initializes the system and sets up the 1394 network stack as interrupt driven code. Along with the required CSR registers [10], the 1394 stack offers the host complete access to the DSP's data and program for debugging purposes (see Figure 4.5).

Figure 4.5 Second Generation System 1394 Memory Map



The system then monitors the command register for changes that the host will make through 1394 asynchronous transactions (see Figure 4.6).

Figure 4.6 Command Register Bits

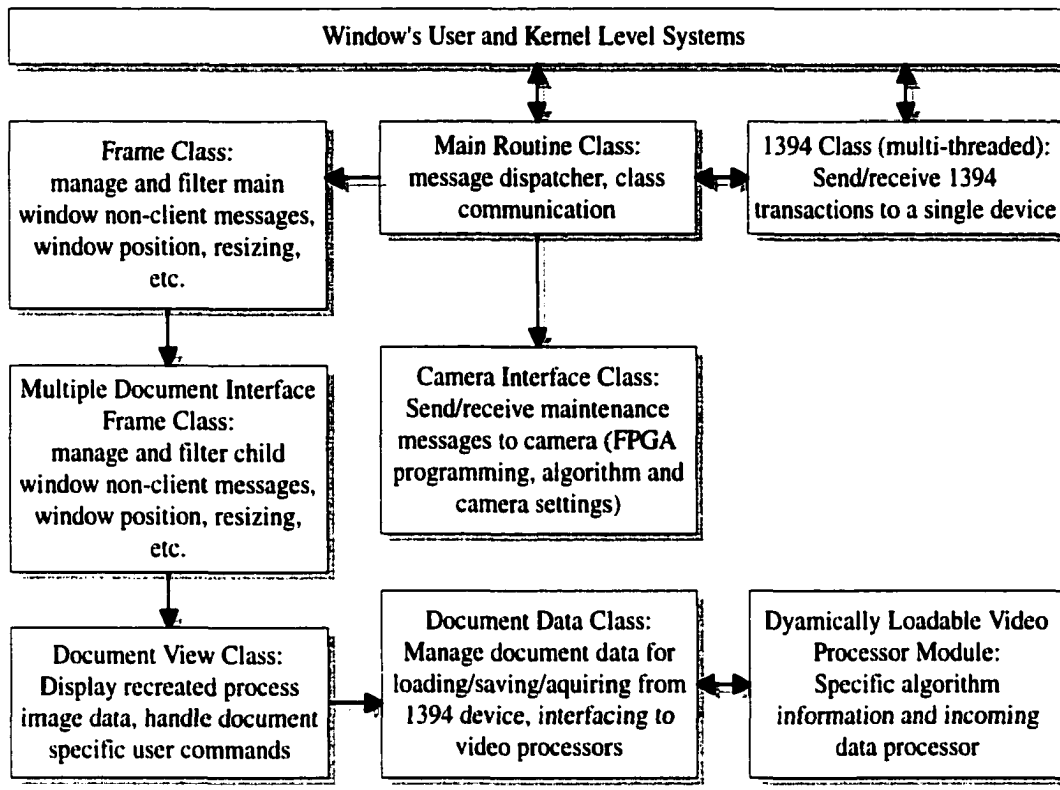


The command register is interpreted by bit decomposition and specific commands are initiated such as programming the FPGA, starting capture, stopping capture, etc. This format closely matches the standardized methods for communicating to low-end frame video cameras [1]. All DSP Code can be found in Appendix D.1 "DSP Code" on page 324.

4.6.2 PC Monitoring/Processing System

The PC software is written in C++ using a series of classes to provide both a graphical user interface (GUI) and an easy coding method to communicate with one or more cameras. The 1394 device interface is separated into a single class which allows for easy upgrading from the Adaptec API to the Microsoft API when fully completed (see Figure 4.7).

Figure 4.7 PC Monitoring/Processing System C++ Classes



Incoming processed data from the camera(s) is received by the kernel to the 1394 thread and placed in the message queue with other Windows messages and are processed when the application receives processor time. Windows messages are processed by the GUI portion of the application while incoming data messages are processed by an external video processing module in the form of a Windows dynamic link library (DLL). This video processor module contains code to control hardware/software algorithm settings, processing the incoming data, algorithm specific DSP processing code and FPGA bitstream. This method allows any algorithm to be added without having to modify the existing application. All PC Windows code can be found in Appendix D.5 "Main PC Host Software Code" on page 457.

4.7 Summary

All the PCBs were tested at each processing step and found to operate as expected with only a few minor changes necessary (resistor, capacitor adjustment and one trace added).

The DSP hardware interface to the FIFOs, 1394 LLC and DALSA MCU also functioned properly. Some changes to the original CPLD logic code was needed to adjust for device timing and errors. The DSP read/write serial communication to the FPGA's register and RAMs operates properly. The DSP 1394 network stack is also operational supporting all 1394 transaction types, however, the host system's interface card drivers have some difficulty communicating to the system during bus resets. A simple algorithm is implemented showing data read from the camera being transmitted to the host system.

Chapter 5

Operational Tests and Results

5.1 Introduction

This chapter presents results of the system test. This follows from the hardware assembly and writing of software and hardware code.

5.2 Test Setup

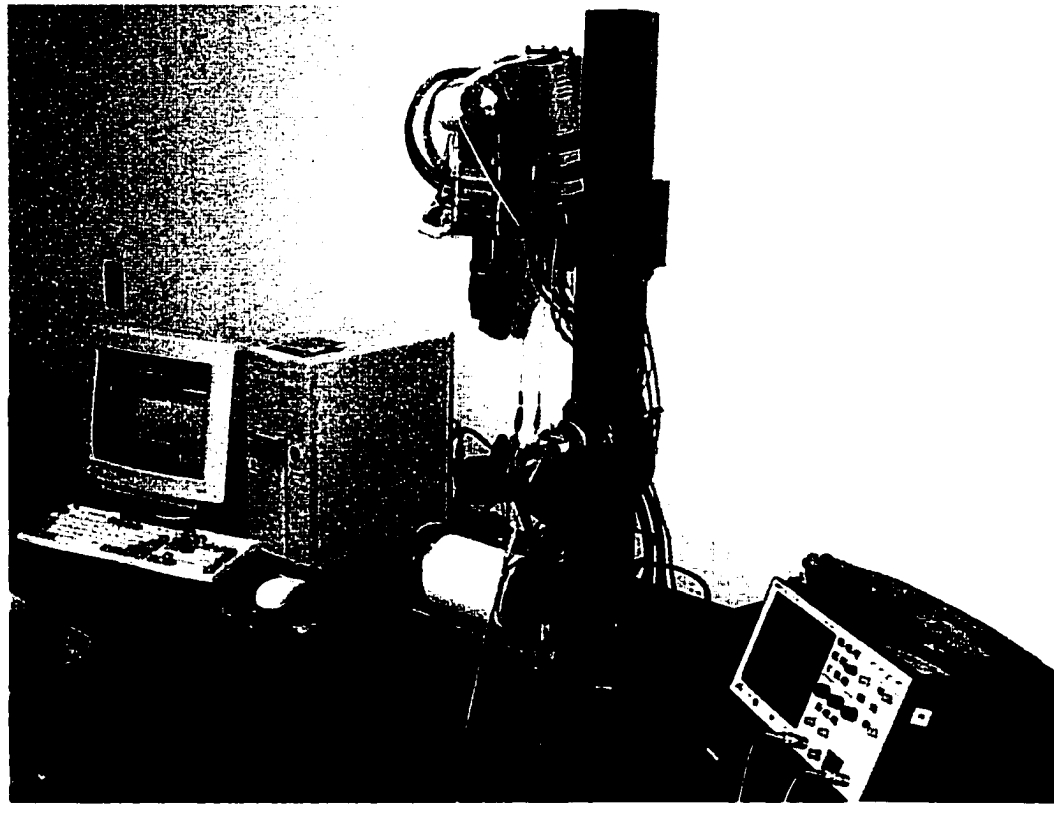
The second generation processing system is assembled and connected to all the debugging sub-systems and mounted within the DALSA camera. A IBM compatible PC running Windows NT v4.0 with a third party 1394 interface card is used as the host/monitoring system for the system. An oscilloscope is also employed to provide useful hardware debugging information (see Figure 5.1).

A miniature conveyor system (a cylinder providing 4096 scan lines per revolution) is used to simulate a manufacturer's production line. Since the conveyor is not on the 1394 bus, the synchronization signals from the shaft encoder are connected directly into the camera processing system.

A high precision power supply (low peak-to-peak noise) providing +/- 5V and +/- 15V is necessary for the camera's highly sensitive

CCD electronics. Because of the transfer characteristics of the isolating DC-to-DC converter on the 1394 board, a standard PC power supply is used to power the 1394 bus.

Figure 5.1 Testing Environment



The PC host system connects to the system primarily with a 1394 cable to support the real-time data transfer operation. For debugging and development purposes, a Xilinx XChecker and an EEPROM emulator are also connected (see Figure 5.2). However, in normal operation only the 1394 cable and power lines would be connected to the system. The XChecker is connected to a 6-pin vertical bus on the DSP board to allow for in-system programming of the CPLDs. The EEPROM emulator is connected directly to the EEPROM socket (using a special male connector) on the DSP board to simplify firmware development and testing without removing, reprogramming and replacing an actual EEPROM device.

Figure 5.2 Test Fixture

5.3 Operational Overview

All firmware code is developed on the PC where it is downloaded to the emulator when the camera system is powered down. Upon power up, the DSP loads its configuration/code from the emulator and is ready for normal operation.

Starting the host/monitoring Windows application on the PC locates the camera on the 1394 bus and waits for the controlling user to choose a video processing algorithm. Parameters for the camera and algorithm can then be set for the particular material under test. Once all parameters are set and the material is in place, real-time processing can begin. The host transmits the video processor specific FPGA bitstream and DSP code to the camera system where the FPGA is programmed and loaded with operational parameters. The programmed FPGA is then instructed to begin processing the real-time data flow from the camera. The FPGA moves this processed data to the FIFO where it is received by the DSP and processed again (if necessary). The data is transferred to the host/ monitoring system using the 1394 isochronous transactions where it is processed again (if necessary) and displayed to the controller. In a real world application, the detection of a variety of defects would result in appropriate product or manufacturing system repairs/ maintenance or improvements.

5.4 Realizing a video processing algorithm

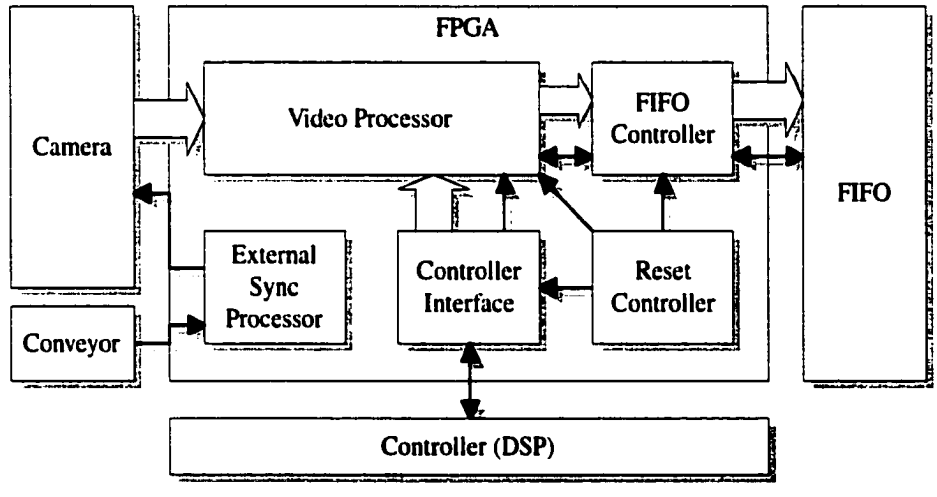
The second generations hardware constraints must be thoroughly considered prior to designing and implementing a new video processing algorithm. The limitation of RAM elements and arithmetic functions in the FPGA force the planning of resource partitioning between the FPGA, DSP and host/monitoring portions of the system. Specifically within the FPGA, the video processing algorithm must interface and co-exist with other internal sub-systems to achieve a functional result.

5.4.1 Other FPGA Sub-Systems

Although the FPGA's main function is to accommodate the video processing algorithm, it also contains the following sub-systems that are crucial to system operation (see Figure 5.3):

- Controller interface (controls sub-systems, receives programmable values for video processing algorithms)
- FIFO controller (controls writes, resets and programmable almost full and empty flags in FIFO)
- Reset controller (performs a stabilized synchronous resets on all sub-systems after programming)
- External sync processor (modifies external synchronizations signals from conveyor appropriately for the camera)

Figure 5.3 FPGA Sub-Systems



Once the in-camera processing algorithm meets all the necessary requirements of the sub-systems, physical implementation can begin.

5.4.2 Algorithm Coding Steps

The steps taken in coding a video processing algorithm for the second generation system are:

1. Write/modify Video Processor VHDL code
2. Analyze with Synopsys Design Compiler; if failed, go to step 1 (fix errors)
3. Link into other FPGA sub-systems code
4. Compile to target Xilinx 4000 series technology
5. Check approximate speed requirements and resource allocation; if failed, go to step 1 (fix to meet constraints)
6. Convert to standard logic gates (Synopsys cannot export directly to 4000 series bit-streams)
7. Export to Xilinx hierarchal design format
8. Optimize logic design with Xilinx optimizer tool (remove extra buffers, etc.)
9. Execute Xilinx partition, place and route tool
10. Check actual speed requirements and resource allocation; if failed, go to step 1 (redesign to meet constraints)
11. Generate downloadable bitstream file
12. Code TMS320C5x DSP code to process data from FPGA system
13. Code video processor Windows DLL with specific decompressing/processing code to interface with monitor/processing application
14. Link in FPGA bitstream and DSP code as resources into video processor DLL module
15. Test algorithm functionality and performance; if failed, go to step 1, 11 or 12

5.5 FPGA Performance

The FPGA in the second generation system is a Xilinx 4010E-3, with 400 CLBs and a CLB delay time of 3ns. It was not possible to obtain a faster device from Xilinx; therefore, the operational speed of the FPGA logic is below the maximum predicted speed.

5.5.1 Video Processing Algorithms

For testing and debugging purposes, the variable camera speed is set to 5MHz. Each algorithm presented here is executed over a span of 8 hours to test the stability of the system. The hardware descriptions of these algorithms can be found in Appendix D.2.4 "Video Processor VHDL Code" on page 407.

Focus Processor

Unlike the first generation system, where real time data transfer was not possible, in this second generation system an uncompressed data stream is able to be moved from the camera to the host system without any data loss. A major use of the transmission of the uncompressed data stream is to allow the set-up procedures associated with focusing the camera lens on the target scanning material, and in adjusting the illumination of the material.

Lineup Processor

The lineup processor passes only the video data on the borders of the CCD sensor. This processor is designed for multiple camera environments where limited video data is needed to set up all the camera positions.

DeltaTracker and Minimum/Maximum Processors

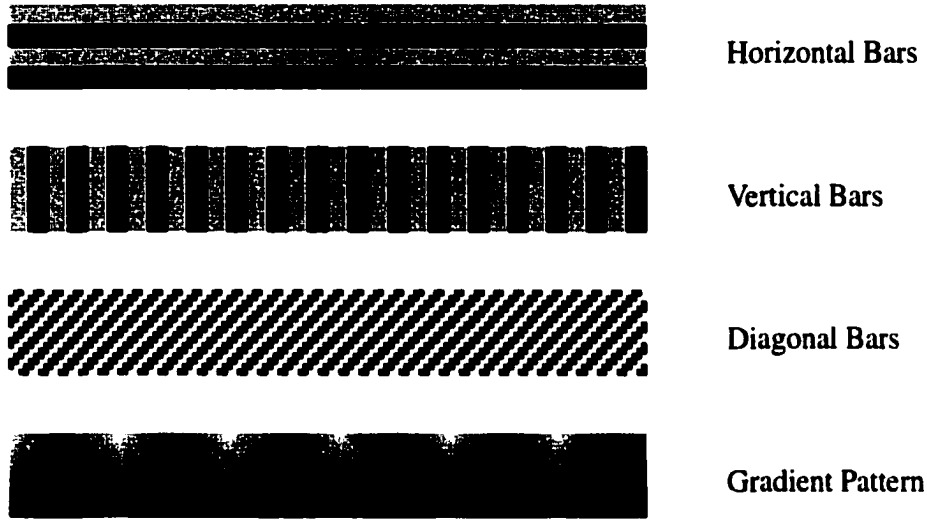
Two of the algorithms implemented in the first generation system have been easily ported to the second generation system for comparison purposes. Both the DeltaTracker and Minimum/Maximum algorithms operate continuously and accurately in this system.

Fuzzy Logic Processor

Based on research work performed by Hossain Hajimowlana [8], a new algorithm, which requires real-time data look-ups, is also implemented. This algorithm is based on a fuzzy

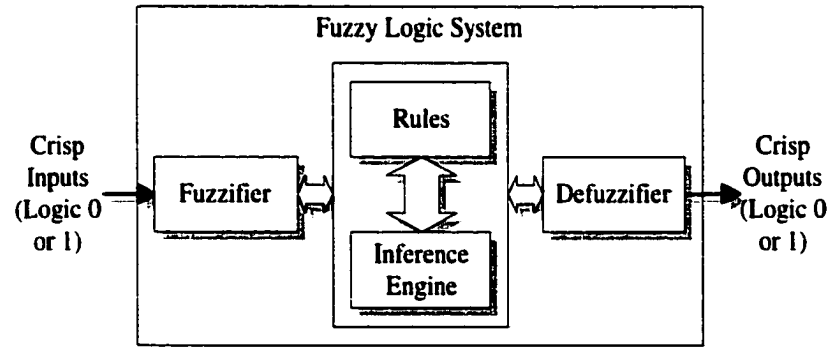
logic paradigm, and can detect defects on patterned backgrounds where other simple algorithms fail (see Figure 5.4).

Figure 5.4 Examples of Patterned Backgrounds



A fuzzy logic system uses a fuzzifier to convert a set of logical inputs into linguistic descriptions such as *moderate* or *very high*. Based on a series of rules, an inference engine interprets these series of descriptions into a final term, for example, *not likely* or *definitely*. A defuzzifier reforms the term into logic outputs to be processed further if necessary (see Figure 5.5).

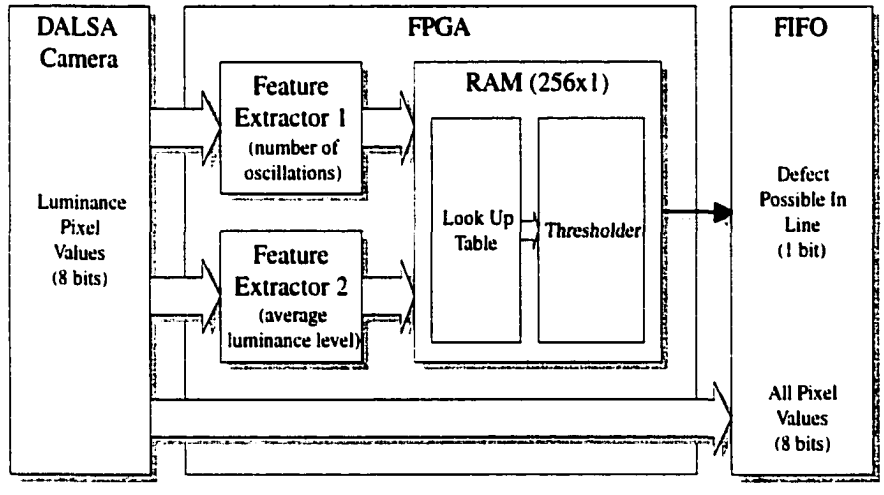
Figure 5.5 Block Diagram of a Fuzzy Logic System



To implement a fuzzy logic system for this application in an FPGA, the crisp inputs (logical) for the fuzzifer are first derived from the input video stream. Through exhaustive

experimentation with unprocessed video samples, correlations between the oscillations and mean of the luminance level were found in a scan lines with defects. Therefore, for this algorithm, two feature extractors are implemented to analyze the number of oscillations in the luminance level (based on a programmable tolerance) and the mean luminance level of a scan line. The complete fuzzy system (fuzzifer, inference engine and defuzzifier) are functionally combined into a single look-up table which is addressed by the combined output of the feature extractors. This simplifies the hardware design constraints and optimizes the allocation of FPGA resources. By analyzing unprocessed video data with off-line software, the contents of this look-up table are computed prior to real-time processing (see Figure 5.6).

Figure 5.6 Fuzzy Component Level Diagram in FPGA



The intent of the fuzzy algorithm is to tag lines with potential defects for further processing by another algorithm that will remove the non-defective background patterns. A DSP based algorithm is also presented that uses an auto-regressive (AR) filter to predict the next pixel values of the video stream. If the pixel value deviates by a programmable tolerance, that pixel value is considered defective. A lookup table is also included to maintain the predictors consistency after a defect is found (see Figure 5.7). Similarly as in the fuzzy algorithm, the coefficients of the AR filter must be computed off-line with unprocessed video samples.

Figure 5.7 Block Diagram of the AR Predictor

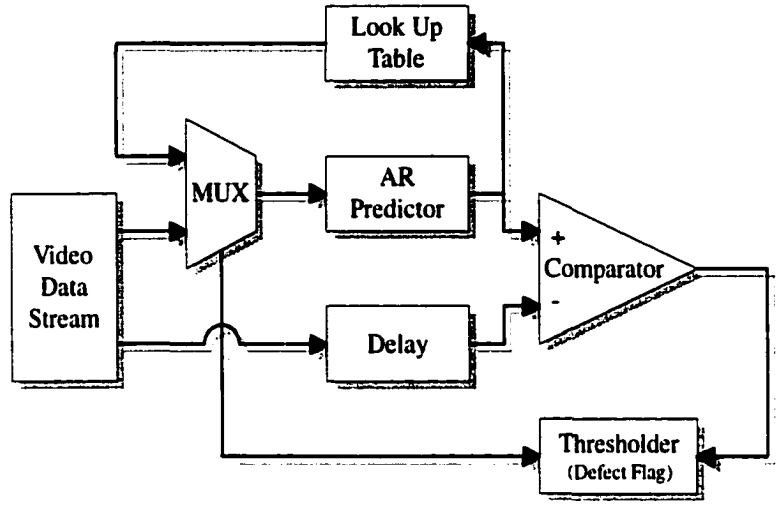


Figure 5.8 shows the monitoring/processing software operating with a series of vertical black and white lines (extremes on the luminance scale) with some defects added. The fuzzy algorithm (implemented in the FPGA and monitoring/processing software only) shows the lines where potential defects are present on the material under test. The AR predictor portion of the system is not implemented in the DSP due to the 1394 system overhead (see "DSP Performance" on page 85), however, examples from [8] are shown in Figure 5.9.

Figure 5.8 Monitoring/Processing System Operating in Real Time

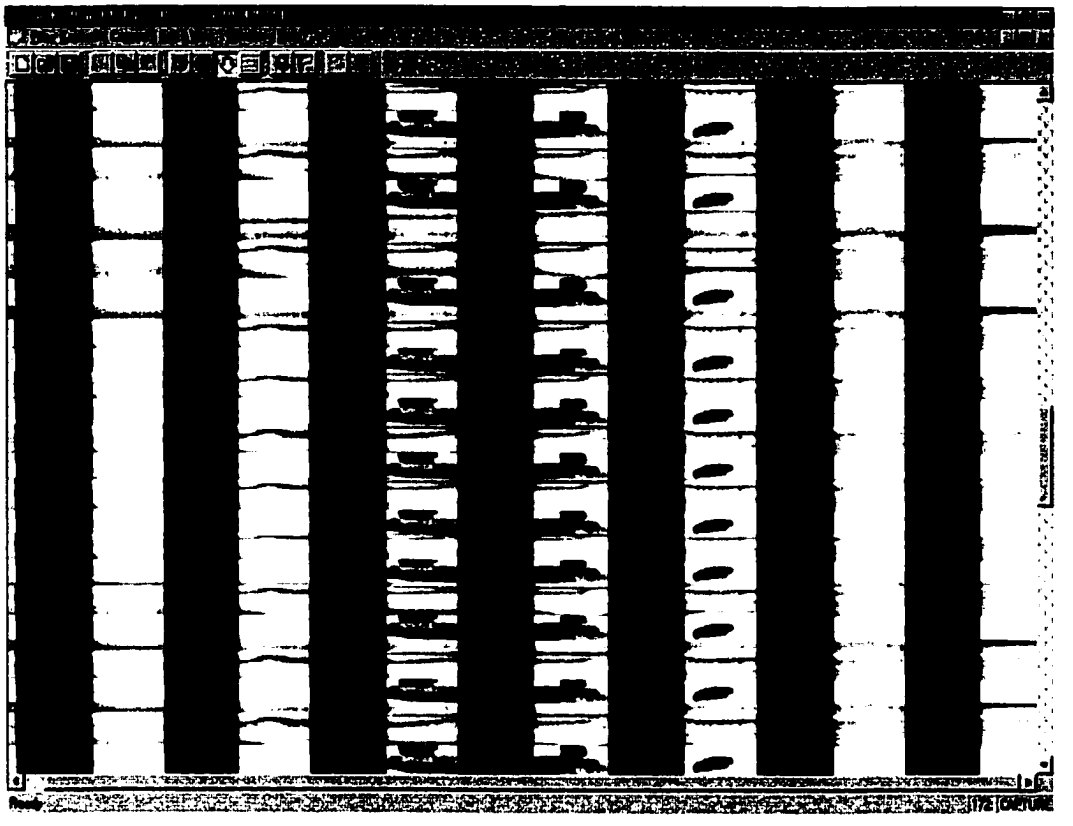


Figure 5.9 AR Predictor Examples

5.5.2 FIFO Performance

Since the FIFOs are synchronous devices, write speeds to the device are limited to the speed of the FPGA (in this case 5MHz). The combined data width of the FIFOs is 27 bits at 5MHz which translates to 135Mbps or 17MBps write throughput. Newer devices

compatible with these older FIFOs are now available in smaller packages with larger depths and data widths [4].

5.5.3 FPGA-to-DSP communication

The FPGA-to-DSP communication operates quickly and reliably, as expected, in order to transmit the register and RAM data prior to processing.

5.6 DSP Performance

The DSP chip did not perform as well as expected, mainly due to the fact that it performs two functions: system controller and arithmetic processor. The 1394 network stack consumed most of the code and most of the processing time in interfacing to the delayed 1394 LLC. Isochronous writes on the 1394 bus were considerably slower than expected only reaching approximately 4MBps with no other DSP processes active.

Performance was also lower since the firmware was loaded into external memory rather than internal memory (which was reserved for system variables and 1394 data). External memory access slows down the DSP since it cannot read/write to data and program memory simultaneously. By dedicating the internal memory to the most sensitive variables and procedures, performance may be increased.

The DSP is only operating at 25MHz (for hardware debugging purposes) but may operate up to a maximum of 33MHz due to limitations with the 1394 LLC. Performance can be increased approximately 33% by increasing the clock speed to maximum.

5.7 1394 Performance

As mentioned above, the interface to the 1394 LLC is unpredictably delayed (see "TSB12C01A Interfacing" on page 50). PHY lockups occur during bus reset events due to an error in the PHY circuity. This was later corrected by adding some passive devices;

however, the PHY no longer operates in repeater mode when the LLC is powered down. These devices are no longer recommended by TI for new designs since newer devices, which correct these problems, are being released. These new devices are also compliant to IEEE-1394A which supports proper transfer at 400Mbps and implements the TI galvanic isolation method internally without external bus holding circuitry [29] [33].

5.8 CPLD Devices

The CPLD devices operated as expected and were very easy to program when soldered within the system. The logic to drive the DSP interface to the 1394 devices could have been improved if more I/O pins were available to introduce background writes. Even in their simplest form these writes could not be implemented since the address lines to the 1394 LLC were not buffered. Because of this the DSP is required to wait until the whole operation to the LLC is completed.

5.9 Summary

Algorithms presented in the earlier chapters and the fuzzy algorithm (briefly described in this chapter) are implemented successfully in the second generation hardware, with the exception of the AR predictor. Although some portions of the system can be improved by using newer components and devices available at the present time, the system design concept is proven.

Chapter 6

Conclusions

6.1 Summary and Contributions

In this thesis we have described the design of a new in-camera processing system, targeted to web defect inspection applications. The processor system is mounted inside a line-scan TDI camera (supplied by DALSA Inc.) and contains three sub-systems: a Field Programmable Gate Array (FPGA) for direct video processing (connected to the digital video bus); a Digital Signal Processor (DSP) chip for system control and off-line processing; an IEEE 1394-1995 networking interface including protocol handling and galvanic isolation for multiple power supplies on the network.

This in-camera processing system represents a second generation design; the first generation design used only a FPGA for the processing hardware and there were no networking capabilities

The in-camera (line scan) processing system can process the video stream much more efficiently in a multiple camera system than using an external analysis system (host PC based), which is the standard approach for most industrial inspection applications. With the addition of the DSP chip, more types of video processing

algorithms are available. The 1394 interface allows an inexpensive networking system to be implemented targeted to multiple camera environments.

Details of the new system design are presented in this thesis and a complete illustrative example is provided using a fuzzy logic algorithm developed by Hossain Hajimowlana in a complementary research project [8].

The software and programmable hardware for the first generation was successfully written to form a basis for the second generation system. The second generation was successfully designed and implemented within the constraints of the target DALSA camera. All of the sub-systems software and programmable hardware operated as expected to prove the design concept.

6.2 Suggestions for Future Work

DALSA's CCD sensors and cameras scanning speeds are increasing rapidly each year. Although the devices used in the second generation system function together to prove the system concept, they will not be fast enough for the latest generation sensor components. Faster devices are needed which can support the real-time requirements as these sensor speeds increase.

Xilinx offers two new series of FPGAs known as the Spartan [36] (low-cost) and the Virtex [37] (high-density) which are based heavily on the 4000E architecture. These devices resolve some of the capacity problems associated with the 4000E series and are projected for larger and faster designs. Accompanying these devices are a new series of libraries and tools which link into the current tool set to provide PCI core interfaces (compliant to v2.1) and DSP core ALUs. These devices could be used to improve real-time FPGA algorithms and provide standardized bus communication within the camera to other devices.

TI has introduced a new level of DSPs, the TMS320C6x series. These new DSPs operate up to 300MHz and can perform 8 instructions per cycle by using very long instruction word (VLIW) architectures. The merging of DSP and VLIW architectures into a single system generates performance characteristics greater than a Pentium level processor. This new DSP is available in two models (fixed point [25] and floating point [26]) which are pin-for-pin compatible. TI has addressed the external memory and device interfacing issues by implementing glueless interfaces to external memory and other commonly used devices [24] (FIFOs, PCI bus, asynchronous bus). Based on the speed of the camera and sensor, multiple TMS320C6x devices could be used to implement sophisticated internal post processing algorithms.

TI has also introduced a new series of high-performance 1394 LLCs for PCI buses. These new LLCs provide internal command FIFOs up to 16K bytes which reduces the PCI bus transfer delays and latencies [32]. Along with the 400Mbps PHY devices [33], the 1394 interface speeds and network performance could be increased to reduce controller overhead.

REFERENCES

[1] 1394 Trade Association, "1394-based Digital Camera Specification", Version 1.04, August 1996; http://www.1394ta.org/Download/Technology/Specifications/Camera104.pdf.

[2] Adaptec, "1394 Developers Kits", March 1997.

[3] Cypress, "CY7C4271 Data Sheet", March 1997; http://www.cypress.com/pub/datasheets/cy7c4261.pdf.

[4] Cypress, "CY7C4285 Data Sheet", November 1997; http://www.cypress.com/pub/datasheets/cy7c4285.pdf.

[5] Dalsa Inc., "Cl-E1/F2 Camera User's Manual", April 1994.

[6] Dalsa Inc., "ML-C3-xxxxK/L Camera User's Manual", February 1997.

[7] Datel, "UWR-12/250-D12 Data Sheet", August 1999; http://www.datel.com/data/power/uwr3w.pdf.

[8] Hossain Hajimowlana, "Efficient Algorithms for a New Design Environment for Defect Detection in Web Inspection Systems", PhD Disertation, University of Windsor, 1999.

[9] IEEE, "P1394 Standard for a High Performance Serial Bus", Draft 8.0v3, October 1995.

[10] ISO/IEC 13213, "ANSI/IEEE Std 1212", 1994 Edition.

[11] Linear Technology, "LT1117 Data Sheet", 1993; http://www.linear.com/pdf/lt1117.pdf.

[12] Linear Technology, "LT1121 Data Sheet", 1994; http://www.linear.com/pdf/lt1121.pdf.

[13] Motorola, "M68HC11 Reference Manual Rev 3.0", 1996; http://mot-sps.com/mcu/documentation/pdf/hc11rmr3.pdf.

[14] Motorola, "MC68HC11 E Series Technical Data Rev. 1", 1995; http://mot-sps.com/mcu/documentation/pdf/hc11er1.pdf.

[15] QiuPing Li, Graham A. Jullien, Jim Roberts, S. Rose, B. Doody, "An In-Camera FPGA system for Video-Stream Processing", Internal Report, VLSI Research Group, University of Windsor, 1995.

[16] Roberts et al., "High Speed Defect Detection Apparatus having Defect Detection Circuits Mounted in the Camera Housing", U.S. Patent Number 5,440,648, August 1995.

[17] Samsung, "KM681002B Data Sheet", June 1997 ; http://www.usa.samsungsemi.com/products/prodspec/speedsram/KM681002B.PDF.

[18] Texas Instruments, "SN74ABTH32316 Data Sheet", SCBS179E, May 1997; http://www-s.ti.com/sc/psheets/scbs179e/scbs179e.pdf.

[19] Texas Instruments, "SN74ACT1071 Data Sheet", SCAS192-D3994, April 1993; http://www-s.ti.com/sc/psheets/scas192/scas192.pdf.

[20] Texas Instruments, "TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide", SPRU018D, March 1995; http://www-s.ti.com/sc/psheets/spru018d/spru018d.pdf.

[21] Texas Instruments, "TMS320C1x/C2x/C2xx/C5x Optimizing C Compiler User's Guide", SPRU024D, March 1995; http://www-s.ti.com/sc/psheets/spru024d/spru024d.pdf.

[22] Texas Instruments, "TMS320C5x DSPs Data Sheet", SPRA030A, April 1996; sprs030a.pdf.

[23] Texas Instruments, "TMS320C5x User's Guide", SPRU056C, January 1997; http://www-s.ti.com/sc/psheets/spru056d/spru056c.pdf.

[24] Texas Instruments, "TMS320C6000 Peripherals Guide", SPRU190C, April1999; http://www-s.ti.com/sc/psheets/spru190c/spru190c.pdf.

[25] Texas Instruments, "TMS320C6201 DSPs Data Sheet", SPRS051F, August 1999; http://www-s.ti.com/sc/psheets/sprs051f/sprs051f.pdf.

[26] Texas Instruments, "TMS320C6701 DSPs Data Sheet", SPRS067C, August 1999; http://www-s.ti.com/sc/psheets/sprs067c/sprs067c.pdf.

[27] Texas Instruments, "TSB12C01A Data Manual", SLLS219A, February 1997; http://www-s.ti.com/sc/psheets/slls219a/slls219a.pdf.

[28] Texas Instruments, "Errata to TSB12C01A", SLLS282, September 1997 ; http://www-s.ti.com/sc/psheets/slls282/slls282.pdf.

[29] Texas Instruments, "TSB12LV01A Data Manual", SLLS332, January 1999; http://www-s.ti.com/sc/psheets/slls332/slls332.pdf.

[30] Texas Instruments, "TSB21LV03A Data Manual", SLLS230A, December 1996; http://www-s.ti.com/sc/psheets/slls230a/slls230a.pdf.

[31] Texas Instruments, "Errata to TSB21LV03A", SLLS281, October 1997 ;http://www-s.ti.com/sc/psheets/slls281/slls281.pdf.

[32] Texas Instruments, "TSB21LV23 Data Manual", SLLS328A, April 1999 ; http://www-s.ti.com/sc/psheets/slls328a/slls328a.pdf.

[33] Texas Instruments, "TSB41LV03A Data Manual", SLLS364, July 1999 ; http://www-s.ti.com/sc/psheets/slls364/slls364.pdf.

[34] Xicore, "X28VC256 EEPROM Data Sheet", April 1996; http://www.xicor.com/PDF_Files/X28VC256.pdf.

[35] Xilinx, "Application Note XAPP065", Version 1.0, July 1996; http://www.xilinx.com/xapp/xapp065.pdf.

[36] Xilinx, "Spartan Series Data Sheet", Version 1.4, January 199; http://www.xilinx.com/partinfo/spartan.pdf.

[37] Xilinx, "Virtex Series Data Sheet", Version 1.6, July 1999; http://www.xilinx.com/partinfo/virtex.pdf.

[38] Xilinx, "XC4000 Series Data Sheet", Version 1.04, September 1996; http://www.xilinx.com/partinfo/4000.pdf.

[39] Xilinx "XC9500 Series Data Sheet", Version 1.1, April 1997; http://www.xilinx.com/partinfo/9500.pdf.

[40] Xilinx "XC9536 Data Sheet", Version 2.0, November 1997; http://www.xilinx.com/partinfo/9536.pdf.

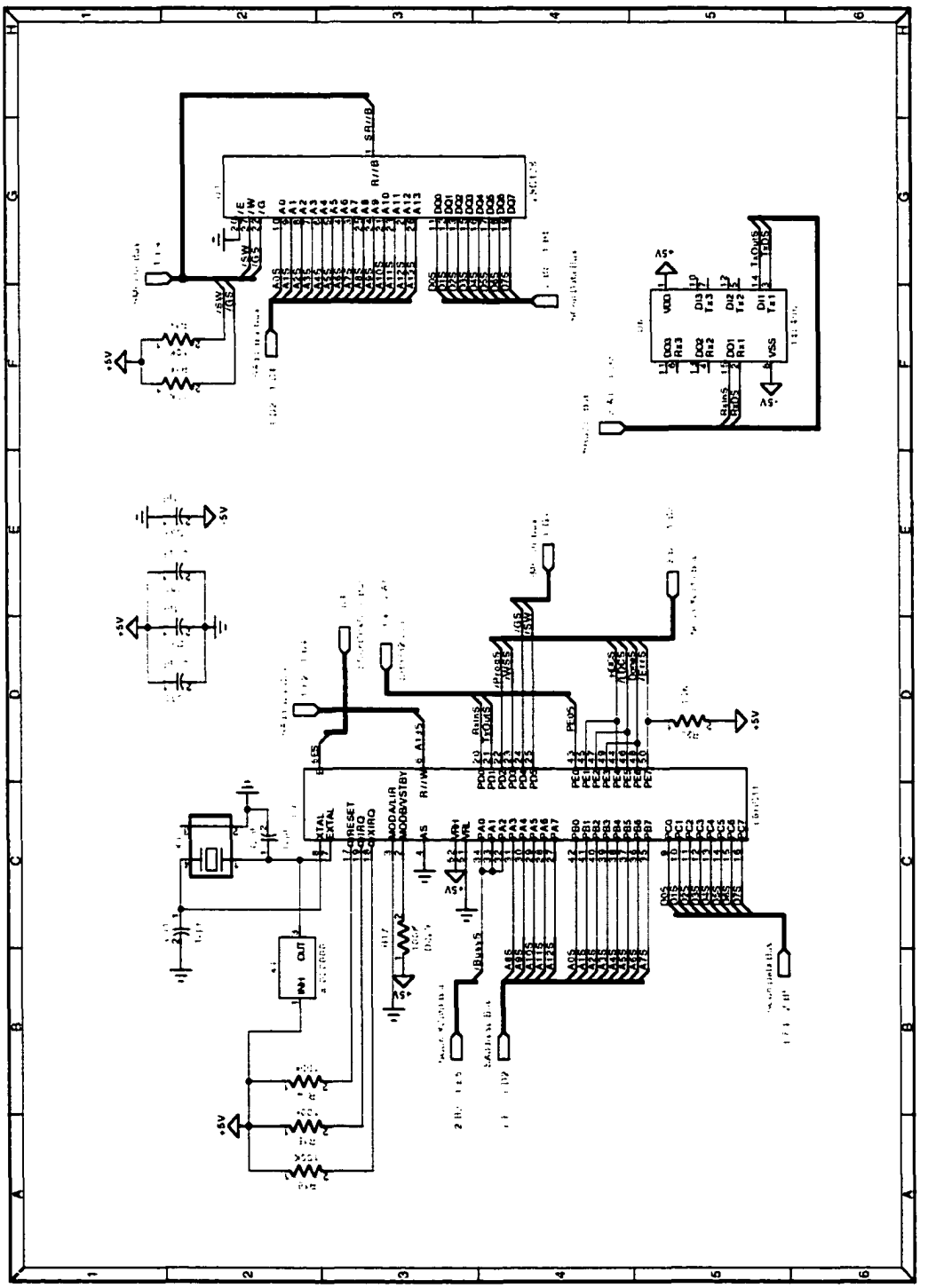
Appendix A

First Generation System Hardware

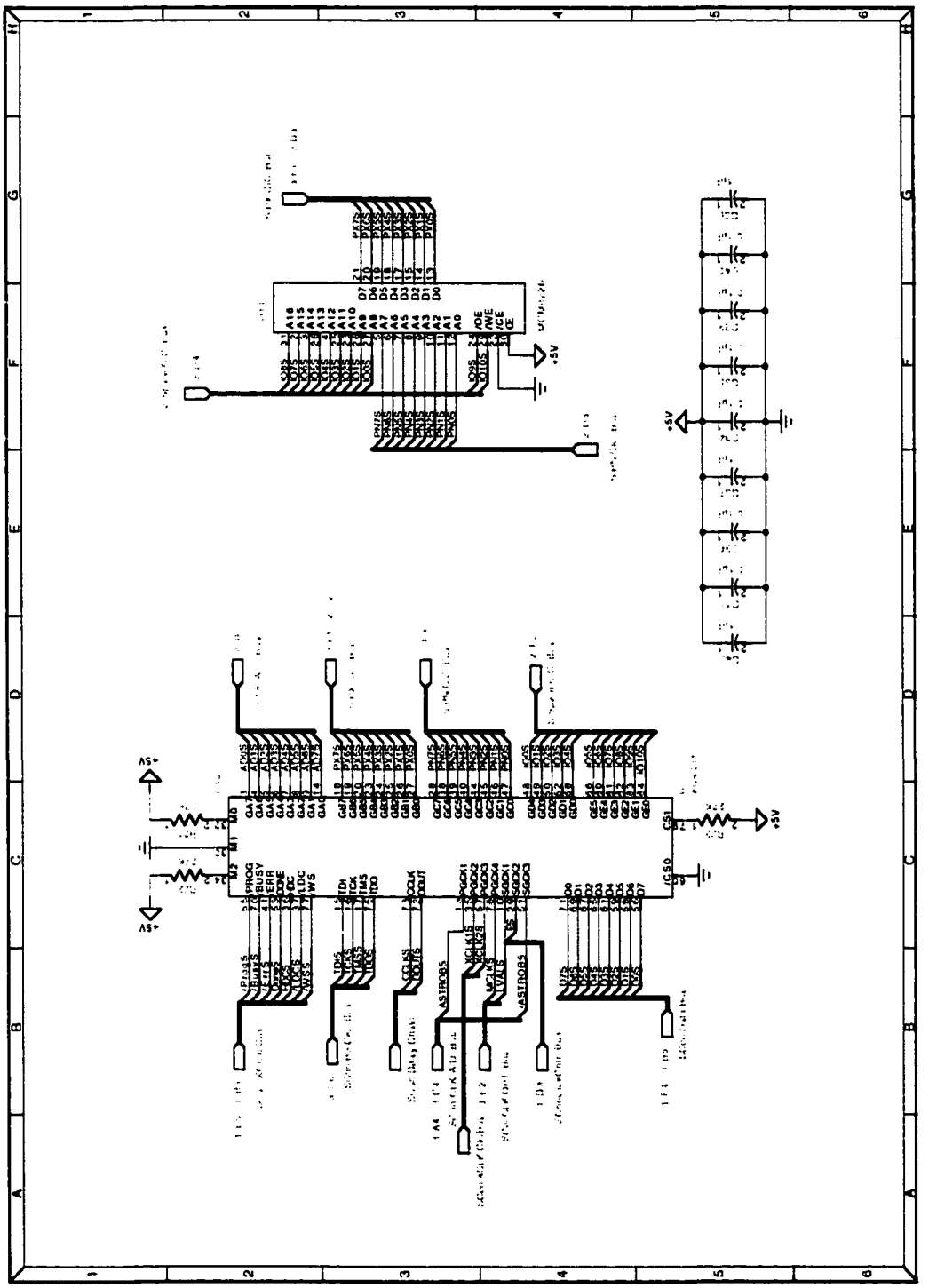
This appendix contains the hardware design for the first generation system. The design itself is in a DesignWorks v3.1.5 schematic capture database (A.1 "Schematics" on page 94) where it is used to verify with the manual layout (A.2 "PCB Layout" on page 97) generated with the Douglas Professional Layout v4.6 tool. The layout is used to manufacture the PCB (A.3 "Fabricated Board" on page 98) by Douglas' PCB Manufacturing Facilities. The assembled board (with the help of Dalsa Inc.) is shown in A.4 "Assembled Board" on page 99.

A.1 Schematics

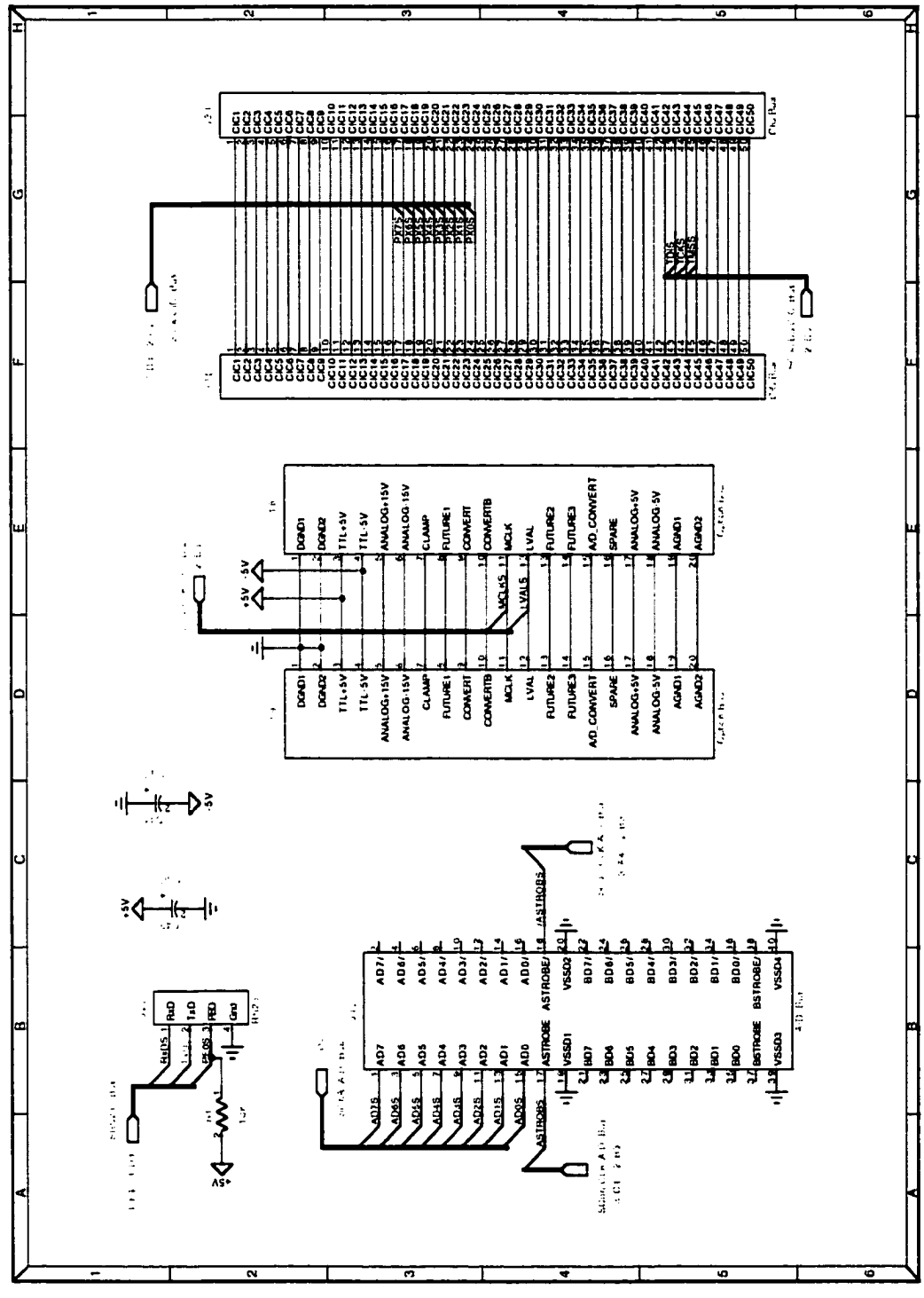
A.1.1 Page 1



A.1.2 Page 2

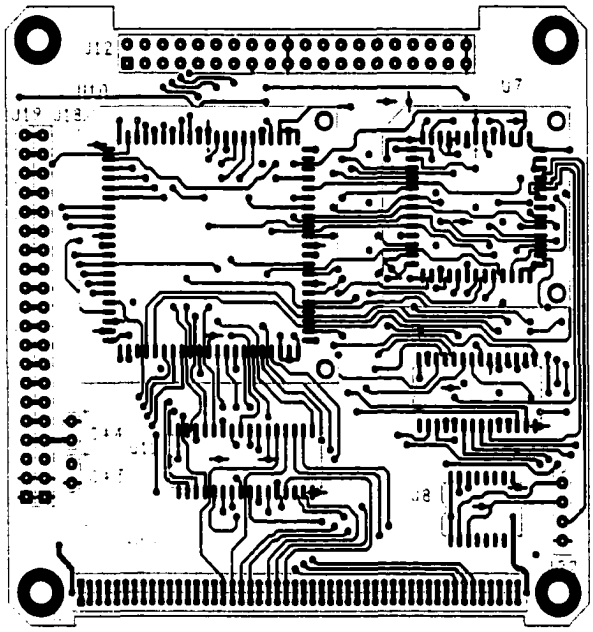


A.1.3 Page 3

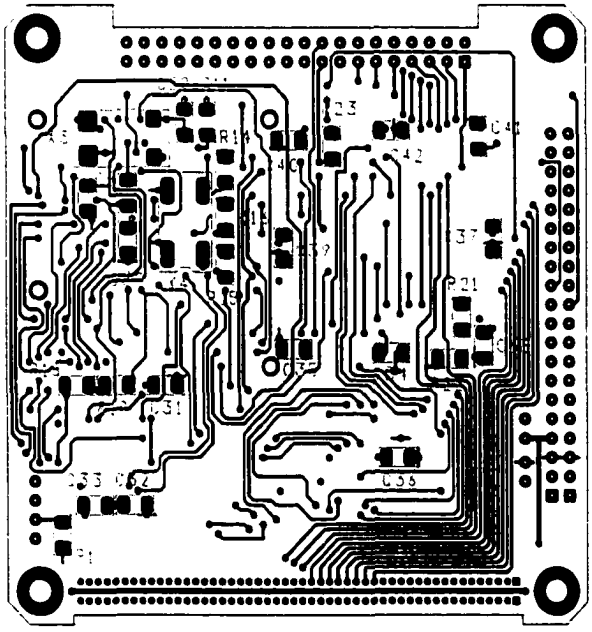


A.2 PCB Layout

A.2.1 Component Side

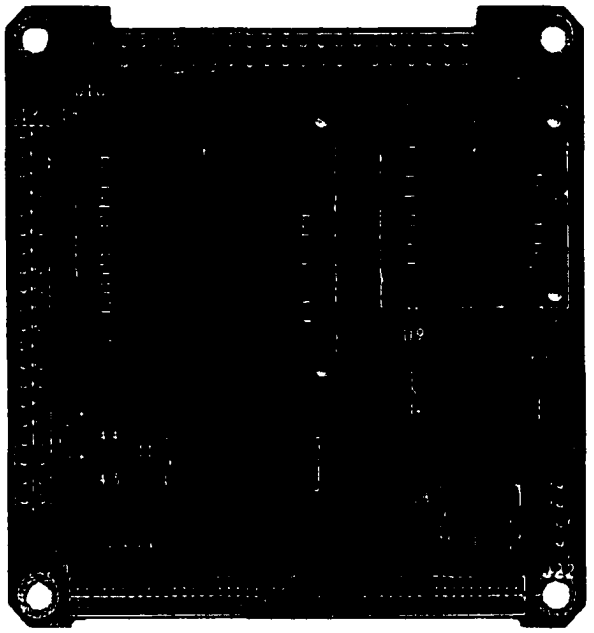


A.2.2 Solder Side

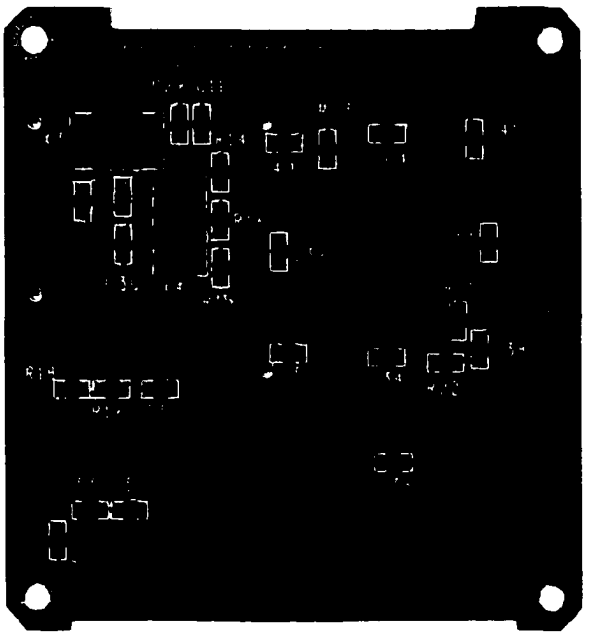


A.3 Fabricated Board

A.3.1 Component Side

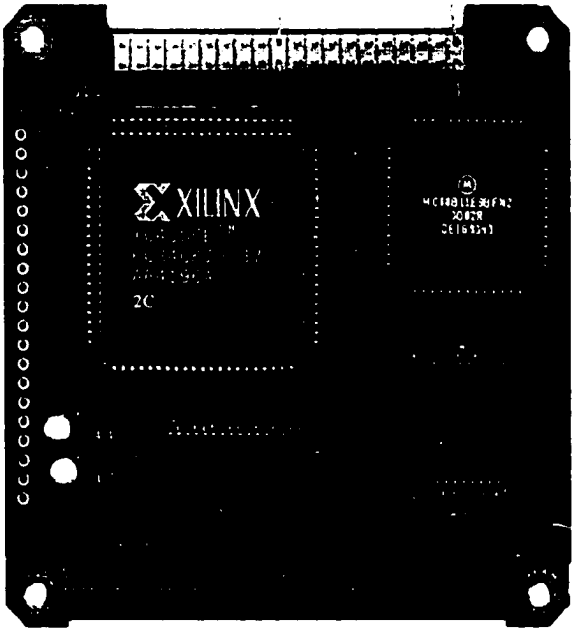


A.3.2 Solder Side

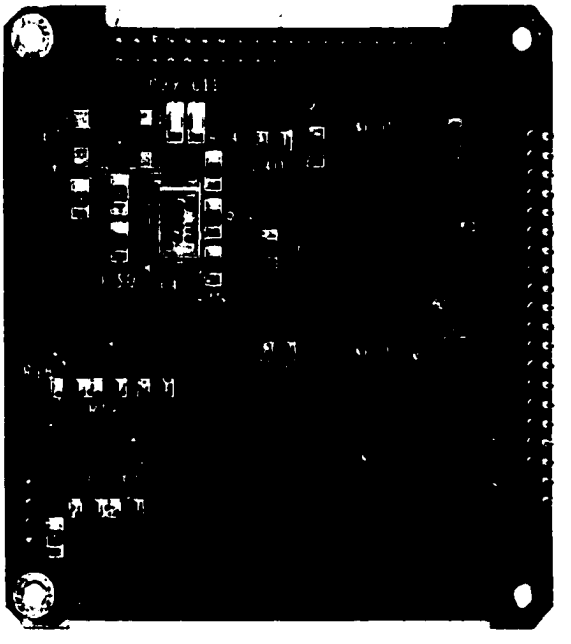


A.4 Assembled Board

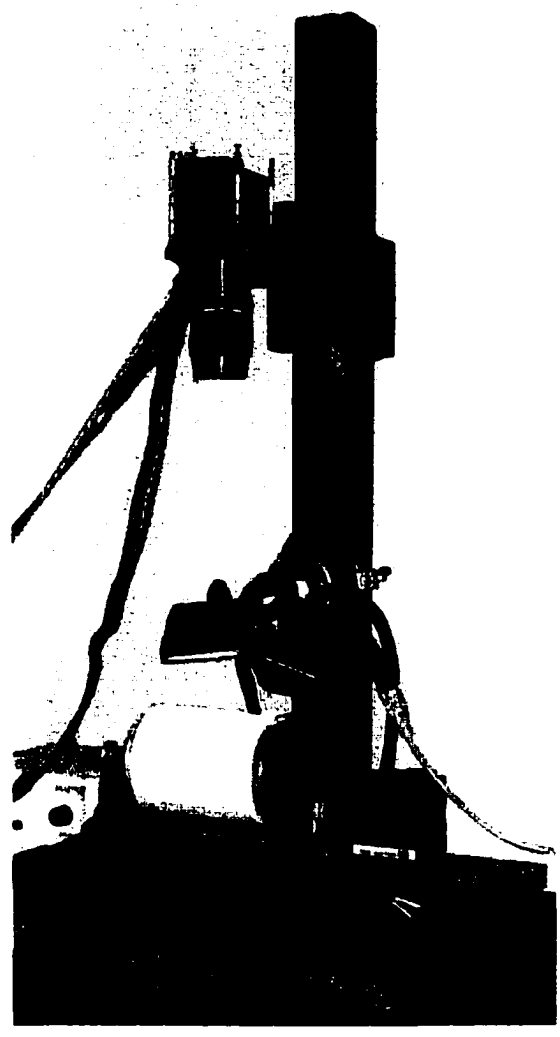
A.4.1 Component Side



A.4.2 Solder Side



A.5 Test Setup



Appendix B

First Generation System Software Code

B.1 MCU Code

B.1.1 Code Building Utilities

The freeware mini-assembler AS11.EXE does not allow for external file referencing ("include" statements). To remedy this, a script is created which attaches the 68HC11 register description file before the desired assembly file and calls AS11 to build the code. For example, to assemble DALSA.ASM, execute: ASM DALSA.ASM. If no errors occur, the final DALSA.S19 is created.

B.1.1.1 ASM.BAT

```
@ECHO OFF
IF NOT EXIST %1.ASM GOTO NOFILE
COPY HC11REG.EQU+%1.ASM TEMP.ASM >> NUL
AS11 TEMP.ASM
IF ERRORLEVEL 1 GOTO ASMERR
IF NOT EXIST %1.S19 GOTO NODELETE
DEL %1.s19 >> NUL
:NODELETE
REN TEMP.S19 %1.S19 >> NUL
GOTO END
:ASMERR
ECHO ASSEMBLY ERROR IN %1.ASM
GOTO END
:NOFILE
ECHO FILE %1.ASM DOES NOT EXIST
:END
IF NOT EXIST TEMP.ASM GOTO STOP
DEL TEMP.ASM >> NUL
:STOP
```

B.1.2 Include Files

B.1.2.1 HC11REG.EQU

```
*
* These equate statements are intended for use with
* the non-multiplexed bus parts of the MC68HC11 family
* of microcontrollers. (e.g. A series, D series, E series, and L series)
*
* They are commonly used as offsets to the x index register
* which contains the register block base address, i.e.
* $1000 for the "A" series, "E" series, and "L" series, and $0000 for the
* "D" series of 6811's
*
porta   EQU     $00
pioc    EQU     $02
portc   EQU     $03
portb   EQU     $04
portcl  EQU     $05
ddrb    EQU     $06
ddrc    EQU     $07
portd   EQU     $08
ddrd    EQU     $09
porte   EQU     $0a
cforc   EQU     $0b
oc1m    EQU     $0c
oc1d    EQU     $0d
tcnt    EQU     $0e
tic1    EQU     $10
tic2    EQU     $12
tic3    EQU     $14
toc1    EQU     $16
toc2    EQU     $18
toc3    EQU     $1a
toc4    EQU     $1c
ti4o5   EQU     $1e
tctl1   EQU     $20
tctl2   EQU     $21
tmsk1   EQU     $22
tflg1   EQU     $23
tmsk2   EQU     $24
tflg2   EQU     $25
pactl   EQU     $26
pacnt   EQU     $27
spcr    EQU     $28
spsr    EQU     $29
spdr    EQU     $2a
baud    EQU     $2b
eprog   EQU     $2b
sccr1   EQU     $2c
sccr2   EQU     $2d
scsr    EQU     $2e
scdr    EQU     $2f
adctl   EQU     $30
adr1    EQU     $31
adr2    EQU     $32
adr3    EQU     $33
adr4    EQU     $34
bprot   EQU     $35
```

```
option  EQU     $39
coprst  EQU     $3a
pprog   EQU     $3b
hprio   EQU     $3c
init    EQU     $3d
test1   EQU     $3e
config  EQU     $3f
*
* the following table of addresses corresponds to the memory
* map of the MC68L11E9; consult a memory map of your device if
* using any other 6811 device in this socket
*
iodev   EQU  $00a5              ;this address is ONLY valid for BUFFALO 3.4
pvic1   EQU  $00e8              ;these other addresses will be constant
pvic2   EQU  $00e5              ; through different versions of BUFFALO
pvic3   EQU  $00e2
initio  EQU  $ffa9
outa    EQU  $ffb8
outcrl  EQU  $ffc4
outstr  EQU  $ffc7
warmst  EQU  $ff7c
buffst  EQU  $e00a
*
ramlow  EQU  $0000
bufstck EQU  $0041              ;this stack is valid for BUFFALO 3.4
rammid  EQU  $0100
ramhi   EQU  $01ff              ;this RAM end is valid for MC68L11E9
regbas  EQU  $1000
eeprom  EQU  $b600
eprom   EQU  $d000
buffalo EQU  $e000
*
```

B.1.3 Internal EEPROM Code

B.1.3.1 DALSA.ASM

DALSA.ASM contains code which is loaded into the MCU's internal EEPROM. This includes the main bootup, and common subroutines for RS-232 and EEPROM functions. The subroutines are placed in an order to optimize the use of the BSR and BRA opcodes (2 bytes) instead of the JSR and JMP opcodes (3 bytes). Due to the slow EEPROM write cycle the DALSA.S19 file should be loaded into the EEPROM using the BUFFALO "load t" instruction at 300 baud.

```
*******************************************************************************
*
* File:  DALSA.ASM
*
* Dalsa Project EEPROM boot software and universal subroutines for
*  the 68HC11 onboard microcontroller
```

```
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: Febuary 15, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

zeroram  EQU  $00
*
loadent  EQU  $c3
tmp1     EQU  $c0
shftreg  EQU  $96
hexbin   EQU  warmst+$09
*
TDELAY   EQU   4000
XON      EQU  $11
XOFF     EQU  $13
*
BOE      EQU  $10
BWE      EQU  $20
BWSS     EQU  $08
BPROG    EQU  $04
*
         org  eeprom
*
* Jump Tables
*
         jmp  main               ; $B600
         jmp  errhand
         jmp  seteeadd
         jmp  geteeadd
         jmp  inceeadd
         jmp  rxnbk
         jmp  rxbk
         jmp  rxbkflow
         jmp  tx
         jmp  txxon
         jmp  txxoff
         jmp  byteread
         jmp  wordread
         jmp  loadprg
         jmp  getptsz
         jmp  bytewrit
         jmp  wordwrit
         jmp  strtload
         jmp  adddiret
         jmp  initport

*
* Error Handler
*

errhand  lda  #$45               ; Right now, just output "ERROR" and jump to
BUFFALO
         bsr  tx
errhand0 jmp  main
```

```
*
* Transmit XON to serial port
*

txxon    ldaa #XON                ; load ACCA with XON code
         bra  tx                  ; transmit

*
* Transmit XOFF to serial port
*

txxoff   ldaa #XOFF               ; load ACCA with XOFF code

*
* Transmit ACCA to serial port
*

tx       brclr scsr,x $80 tx      ; wait for bit 7 of scsr to go high
         staa scdr,x              ; send data
         rts                      ; return

*
* Output a string
*

ostring  jsr  outstr
         ldx  #regbas
         rts

*
* Recieve data from serial port (non-blocking), result in ACCA, Z set if none
*

rxnbk    brset scsr,x $20 rxbk0   ; Check to see if anything is waiting for us
         clra                     ; set Z flag on and ACCA to zero
         rts                      ; Nothing waiting, return
*
* Receive data from serial port (blocking), result in ACCA
*

rxbk     brclr scsr,x $20 rxbk    ; wait for bit 5 of scsr to go high

* Get the serial status register and check it for frame errors, overruns or noise

rxbk0    ldaa scsr,x              ; Check for errors
         anda #$0e
         beq  rxbk1               ; if no errors, get data and return
         ldaa scdr,x
         bra  errhand
*         rora                     ; shift right once
*         ora  #$30                ; or $30 to get a numerical result
*         jsr  outa                ; output transmit error type
*         bra  errhand             ; jump to error handler
rxbk1    ldaa scdr,x              ; get received data
rxbk2    rts                      ; return

initport EQU  *

*
```

```
* Disable some devices on the EVAL board
*

        clra
        tap                     ; Stop mode, xirq, and irq are enabled
        staa iodev              ; Tell BUFFALO that we are using the SCI
        jsr  initio             ; Initialize SCI IO
        ldx  #regbas
        ldaa #$15
        staa hprio,x            ; Turn off the E-clock to save power

*
* Disable SPI and A/D converter
*

        clra
        staa spcr,x             ; Disable SPI
        staa option,x           ; Disable A/D converter system

*
* Set ALL Port C to inputs
*

        clr  ddrc,x             ; Set PORTC to all inputs

*
* Enable Port A [7:3] and Port B [7:0] as a 13 bit output port (EEPROM address
bus)
*

        ldaa #$8c
        staa pactl,x

*
* Enable Port D [3:2] as a 2 bit output port (EEPROM ~Write_Enable and
*  ~Output_Enable)
* Port D [1:0] TxOutS, RxInS
* Port D [3:2] /WSS /ProgS
* Port D [5:4] ~Write_Enable and ~Output_Enable
*
        ldaa #$3c
        staa ddrd,x

*
* EEPROM ~WE high, ~OE high.  XILINX ~PROG high (EEPROM in safe mode, XILINX
* 'in non-programming mode)
*

        ldaa #$30
        staa portd,x

*
* Set FULL-INPUT HANDSHAKE mode
* This will allow us to use PIN 6 as a static output
* This is controlled by the LSB of "pioc", and note that the output is inverted
* NOTE: DO NOT READ PORTCL or pin 6 will toggle unexpectedly
*

        ldaa #$11
```

```
        staa pioc,x

        rts

*
* Receive data from serial port (block) with XON/XOFF, result in ACCA
*

rxbkflow bsr txxon
        bsr rxbk
        psha
        bsr txxoff
        pula
        rts

*
* Main bootup routine
*

main    EQU *

        lds  #bufstck           ;initialize the stack pointer

        clr  regbas+bprot

        bsr  initport

main0   ldaa #$30
        bsr  tx

        bsr  rxbk

        cmpa #$03               ; $3 = buffalo
        bne  main1
        jmp  buffst

main1   cmpa #$20               ; $20 = load and run @ 100
        bne  main2
        jmp  ldrn
main2   blt  main0              ; $21-$2f prog $1-$f
        cmpa #$2f
        bgt  main0
        anda #$0f
        tab
        clra

*
* Load program in EEPROM at index ACCD into microcontroller
*

loadprg  bsr  getptsz           ;get directory entry
        ldd  zeroram            ;move to start pointer
        bsr  seteeadd

loadprg0 bsr  wordread          ;get memory address
        xgdy                    ;move it to the Y
        bsr  wordread           ;get packet size
        cpd  #0                 ;is it zero?
        beq  loadprg2           ;if zero, we are done, execute
```

```
loadprg1 std  zeroram+4          ;if not, save it

         bsr  byteread           ;get data
         staa 0,y                ;write it to ram
         iny                     ;increment the pointer
         sec                     ;subtract 1 from the size
         ldd  zeroram+4
         subd #1
         bne  loadprg1           ;if its not zero get more
         beq  loadprg0           ;when its zero, go to the next packet

loadprg2 jmp  0,y                ;jump to the execution location

*
* Start EEPROM loading procedure, sets the address for program load
*

strtload ldaa #$30
         staa portd,x

         clra
         clrb
         staa tmp1
         staa shftreg+1
         bsr  getptsz
         ldd  zeroram
         bsr  seteeadd

         bsr  rxbkflow
         jsr  hexbin
         ldaa shftreg+1
         staa loadent
         rts

*
* Read a WORD from the EEPROM, result in ACCD
*

wordread bsr  byteread
         psha
         bsr  byteread
         tab
         pula
         rts

*
* Read a BYTE from the EEPROM, result in ACCA
*

byteread bclr portd,x $10
         ldaa portc,x
         psha
         bsr  inceeadd
         pula
         bset portd,x $10
         rts

*
```

```
* Set EEPROM Address in ACCD
*

seteeadd stab portb,x            ; store low order bits (0-7) to port B
         asla                    ; shift higher order bits (8-12) left 3 times
         asla                    ; the last shift will place the MSB on the carry
         asla
         staa porta,x            ; store higher order address to port A
         ldaa #$11
         bcc  inceead2
         deca
         bra  inceead2

*
* Get EEPROM pointer and size in ACCD, returned at zeroram+(0-3)
*

getptsz  asld
         asld
         bsr  seteeadd
         clr  ddrc,x
         bsr  wordread
         std  zeroram
         bsr  wordread
         std  zeroram+2
         rts

*
* Write a WORD to the EEPROM (High (A) -Low (B)) and increment the address
*

wordwrit pshb
         bsr  bytewrit
         pula

*
* Write a BYTE to the EEPROM and increment the address
*

bytewrit bset portd,x $10        ; turn EEPROM read mode off
         bset ddrc,x $ff         ; set port C to output
         staa portc,x            ; place data on bus
         bclr portd,x $20        ; set WE low
         bset portd,x $20        ; set WE high
         pshy                    ; save the Y index
         ldy  #4290              ; wait for write cycle to finish
waitwrit dey
         bne  waitwrit
         clr  ddrc,x             ; set port C to input
         puly                    ; restore the Y and follow through the next
routine

*
* Increment EEPROM Address, ACCD destroyed
*

inceeadd inc  portb,x            ; increment port B (0-7 address bits)
         bne  inceead0           ; if port B has bot incremented to 0, then skip
port A modification
```

```
        ldaa porta,x            ; add 8 to port A (8-12 address bits)
        adda #8
        staa porta,x
inceead1 bcc  inceead0          ; set up for pioc modification (ALWAYS OR BY
$18)
        ldaa pioc,x             ; if the carry is not set, store value anyway
        eora #1                 ; decrement ACCA, remember pin 6 output is
inverted
inceead2 staa pioc,x
inceead0 rts                    ; return

*
* Add a directory entry pointer
*

adddiret bsr  geteeadd
        sec
        subd zeroram
        xgdy
        clra
        clrb
        bsr  getptsz
        ldd  zeroram+2
        std  zeroram+4
        sty  zeroram+2
        ldab loadent
        clra
        bsr  setptsz
        ldd  zeroram
        addd zeroram+2
        std  zeroram
        sec
        ldd  zeroram+4
        subd zeroram+2
        std  zeroram+2
        clra
        clrb

*
* Set EEPROM pointer in ACCD, pointer and size in zeroram+(0-3)
*

setptsz  asld
        asld
        bsr  seteeadd
        ldd  zeroram
        bsr  wordwrit
        ldd  zeroram+2
        bsr  wordwrit
        rts

*
* Get EEPROM Address, result in ACCD
*

geteeadd ldaa pioc,x            ; get value of pin 6 from pioc
        eora #1                 ; invert it
        asra                    ; shift it to the right, which is an easy way
to put it in the carry flag
```

```
        ldaa porta,x            ; get the higher order (8-12) bits from port A
        rora                    ; shift to the right 3 times
        rora
        rora
        anda #255-128-64        ; remove higher order bits (15-14)
        ldab portb,x            ; load ACCB (lower ACCD) with low order bits (0-
7)
        rts                     ; return

*
* Load and RUN prg at $100
*

ldrn    jsr  rxbk
        staa zeroram

        ldy  #$100
ldrn0   jsr  rxbk
        staa 0,y
        iny
        dec  zeroram
        bne  ldrn0

        jmp  $100
```

B.1.4 Downloadable Modules

B.1.4.1 EEFORMAT.ASM

The EEFORMAT code formats the external EEPROM filing system. The first 64 bytes (for 16 entries) contains the pointers and sizes of the files and free space. This code is executed using the $20 command instruction (see "DALSA.ASM" on page 103) which executes incoming MCU code without loading it into the EEPROM filing system.

```
*******************************************************************************
*
* File:  LDEEFPGA.ASM
*
* Dalsa Project EEPROM formatting software for 68HC11 onboard
*  microcontroller
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: March 28, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

main    EQU  eeprom
errhand EQU  main+3
seteeadd EQU  errhand+3
```

```
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
txxon    EQU  tx+3
txxoff   EQU  txxon+3
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3

*
* Some extra equates for this program
*

dirents  EQU  16
ramsize  EQU  16384
dirsize  EQU  dirents*4

*
* EEPROM Memory MAP
*
* Lower 16K memory (Upper 16 for FPGA bit map)
*
* $0000  Location of free space
* $0002  Size of free space
* $0004  Location of file 1
* $0006  Size of file 1
* $0008  Location of file 2
* $000A  Size of file 2
* .....
* $003C  Location of file 15
* $003E  Size of file 15
*

*
* Set program origin to the start of free RAM
*

        ORG  rammid

eeformat EQU  *

*
* Use EEPROM function to initialize the ports
*

        jsr  initport

*
* Write a "G" to the host
*
```

```
*         ldaa #$47
*         jsr  tx

*
* We first place the EEPROM into a safe mode.  The FPGA (after a reset) will
*  have all of its programmable pins set to high impedance.  Its probably not
*  a good idea to run this when the FPGA is programmed with an unknown
*  configuration.
*
* Set /WE high and /OE high on EEPROM
* Set /WS low and /PROG low on FPGA
*
* Make sure /WS is low since it controls the high order address line to the
* external EEPROM.
*

         ldaa #$30
         staa portd,x

*
* Initialize the address of the EEPROM to 0000
*

         ldd  #0
         jsr  seteeadd

*
* Write free pointer
*

         ldd  #dirsize+4
         jsr  wordwrit
         ldd  #ramsize-dirsize-4
         jsr  wordwrit

*
* Write "dirent" directory entries (filled with zeros)
*

         ldy  #dirsize

fillloop ldaa #0
         jsr  bytewrit

         dey
         bne  fillloop

*
* Jump to MAIN
*

         jmp  main
```

B.1.5 External EEPROM Modules

B.1.5.1 LDS19.ASM

The LDS19 code reads in an S19 file from the host and converts it into a segmented code format which can be loaded and executed from the main command level.

```
*******************************************************************************
*
* File:  LDS19.ASM
*
* Dalsa Project S19 loading software into EEPROM filing system for
*  the 68HC11 onboard microcontroller
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: March 12, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

*
* Some BUFFALO Equates
*

prt0     EQU  $ae
prt1     EQU  $b0
prt2     EQU  $b2
prt3     EQU  $b4
prt4     EQU  $b6
prt5     EQU  $b8
prt6     EQU  $ba
prt7     EQU  $bc
prt8     EQU  $be

ldoffst  EQU  $ac
count    EQU  $a8
chrcnt   EQU  $a9

tmp1     EQU  $c0
tmp2     EQU  $c1
tmp3     EQU  $c2

shftreg  EQU  $96

hexbin   EQU  warmst+$09
dchek    EQU  $ffa6
upcase   EQU  $ffa0

*
* EEPROM routines
*

main     EQU  eeprom
```

```
errhand  EQU  main+3
seteeadd EQU  errhand+3
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
txxon    EQU  tx+3
txxoff   EQU  txxon+3
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3

*
* Some extra equates for this program
*

prgstrt  EQU  prt6
written  EQU  prt7
secsize  EQU  prt8

*
* Set program origin to the start of free RAM
*

         ORG  rammid

lds19    EQU  *

*
* Call EEPROM to initialize ports
*

         jsr  initport

         jsr  strtload        ;create a file in EEPROM filing system

         clr  tmp3

getmore  jsr  rxbkflow        ;skip white space
         jsr  dchek
         beq  getmore
         cmpa #$0a
         beq  getmore
         jsr  upcase          ;check for "S"
         cmpa #$53
         bne  lderror         ;error out if it isn't an S

         bsr  rdbegin         ;read header

         tst  tmp2            ;is it the 9 record
         bne  cksum
```

```
        tst  tmp3           ;is this the first line?
        beq  firstln

        cpd  ldoffst        ;not first line, check if continuing from
        beq  mrbytes        ; previous address

        bsr  wreeprom       ;if not flush old size

firstln bsr  inbegin        ;write new pointer (on first line too)

mrbytes jsr  getbyte        ;get data byte
        jsr  bytewrit       ;write it

        ldd  written        ;update bytes written in section
        addd #1
        std  written

        ldd  ldoffst        ;update pointer
        addd #1
        std  ldoffst

        dec  count          ;decrement line counter
        bne  mrbytes

cksum   std  prgstrt        ;verify checksum
        ldab chrcnt
        comb
        bsr  getbyte
        cba
        bne  lderror        ;error out if it fails

        bset tmp3 $1

        brclr tmp2 $ff getmore

        bsr  wreeprom       ;write program beginning
        ldd  prgstrt
        bsr  inbegin

        jsr  finiload       ;update directory entry

        jsr  txxon          ;enable flow control, leave
        jmp  main

lderror jsr  txxon          ;error, enable flow control, goto global error
        jmp  errhand

rdbegin jsr  rxbkflow       ;get second character
        anda #255-49
        staa tmp2           ;set flag it it's a 9

        clr  chrcnt         ;get line length byte and save
        bsr  getbyte
        sec
        suba #3
        staa count
        bsr  getbyte        ;get MCU offset
        bsr  getbyte
```

```
        ldd  shftreg
        rts

wreeprom jsr  geteeadd          ;save write pointer
        psha
        pshb
        ldd  secsize            ;Write size value
        jsr  seteeadd
        ldd  written
        jsr  wordwrit
        pulb
        pula
        jsr  seteeadd           ;return back to previous pointer

        ldd  shftreg

anrts   rts

inbegin  std  ldoffst           ;Write MCU address
        jsr  wordwrit
        jsr  geteeadd           ;rememeber size location
        std  secsize
        clra
        clrb
        std  written            ;write zero size for now
        jmp  wordwrit

getbyte  clr  tmpl              ;Get a hex (2 ascii bytes) from host
        jsr  rxbkflow
        jsr  hexbin             ;use BUFFALO conversion code
        jsr  rxbkflow
        jsr  hexbin
        ldaa shftreg+1
        adda chrcnt
        staa chrcnt
        ldaa shftreg+1
        brclr tmpl $ff anrts
        pula
        pula
        bra  lderror
```

B.1.5.2 LDEEFPGA.ASM

The LDEEFPGA code reads in a FPGA bitstream (stripped of Xilinx file and date fields) and saves it into the upper 16K of the external EEPROM. This code performs burst writes to the EEPROM (up to 64 writes in a single cycle) to improve downloading time for 9600 baud (originally done at 300 baud).

```
*******************************************************************************
*
* File:  LDEEFPGA.ASM
*
* Dalsa Project EEPROM burst loading software for the 68HC11 onboard
```

```
*  microcontroller to load the FPGA bitmaps
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: March 28, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

*
* This program read an FPGA bitmap from the host computer and stores it in
*  the upper 16K of the external EEPROM.  Prior to the FPGA data, 2 bytes
*  contain the number of bytes in the bitmap to follow.
*

*
* BUFFALO Equates
*

prt0     EQU  $ae
prt1     EQU  $b0
prt2     EQU  $b2
prt3     EQU  $b4
prt4     EQU  $b6
prt5     EQU  $b8
prt6     EQU  $ba
prt7     EQU  $bc
prt8     EQU  $be

ldoffst  EQU  $ac
count    EQU  $a8
chrcnt   EQU  $a9

tmp1     EQU  $c0
tmp2     EQU  $c1
tmp3     EQU  $c2

shftreg  EQU  $96

hexbin   EQU  warmst+$09
dchek    EQU  $ffa6
upcase   EQU  $ffa0

*
* EEPROM routines
*

main     EQU  eeprom
errhand  EQU  main+3
seteeadd EQU  errhand+3
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
txxon    EQU  tx+3
txxoff   EQU  txxon+3
```

```
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3

*
* Some extra equates for this program
*

*
* pagesiz is the number of bytes to write in one EEPROM write cycle.  This
*  value should always be a power of 2 and should not exceed 32 since the
*  68HC11 is fast enough in this configuration.  However, only a maximum
*  of 64 bytes can be written in one write cycle since the A6-A14 lines
*  cannot be altered.
*

pagesiz  EQU  32

brbufend EQU  $200
brbufbeg EQU  brbufend-pagesiz

byteslef EQU  $00
totalbyt EQU  $02
flushlim EQU  $04

*
* Set program origin to the start of free RAM
*

         ORG     rammid

ldeefpga EQU  *

*
* Use EEPROM function to initialize the ports
*

         jsr  initport

*
* We first place the EEPROM into a safe mode.  The FPGA (after a reset) will
*  have all of its programmable pins set to high impedance.  Its probably not
*  a good idea to run this when the FPGA is programmed with an unknown
*  configuration.
*
* Set /WE high and /OE high on EEPROM
* Set /WS high and /PROG low on FPGA
*
* Make sure /WS is high since it controls the high order address line to the
* external EEPROM.
*

         ldaa #$38
```

```
        staa portd,x

*
* Set PORT C to outputs
*

        bset ddrc,x $ff

*
* Initialize the address of the EEPROM to 0000
*
        ldd  #0
        jsr  seteeadd

*
* Set buffer pointer
*

        ldy  #brbufbeg

*
* The FPGA code actually contains the number of bytes in the bitmap.  We
*  will process this data to determine when this program should complete.
*
* The first byte of the FPGA bitmap is a $FF or a dummy byte.  It
*  contains no valid information.
*
* The next byte is a combination of both the preamble code and high
*  order size code.  The first 4 bits of this code should be "0010" or
*  $2.  The next for bytes are the high order size of the 24 bit size
*  code.  This code is in terms of bits to be received.
*
* The next two bytes are the middle bits of the size code and the
*  remaining byte is a combination of the low order size code and
*  another dummy code.  4 bits for the low order size and 4 bits "1111"
*  or $f for dummy code.
*
* Example:
*
* <   FF  > <   20        > <   0D  >  <   2D  >  <   6F     > < .....
*
* 1111 1111 0010       0000 0000 1101  0010 1101  0110 1 1 1 1 xxxxxxxx
*
* <dummy  > <preamble> <length count                > <dummy> <data...
* <   bits> <    code> <                             > < bits> <frame..
*
* The length cound is: 00D2D6 = 53974 bits
*
* Dividing this value by 8 we get 6746.75, which must be rounded up
*  to 6747 which is the number of bytes required for the XC4003PC84
*  FPGA.
*
* The data's last byte is a postamble code which should contain
*  "01111111" or $7f.
*


*
* Write two dummy bytes, these will be filled in later with the size
*  of the FPGA data in bytes.  The loader will then know how many bytes
```

```
*  to send to the FPGA initally.
*

        ldaa #0
        bsr  wrbrbyte
        bsr  wrbrbyte

*
* Lets make sure the dummy bits are all ones.
*

        jsr  rxbk
        cmpa #$ff
        bne  lderror

*
* In future versions, the preamble and postamble code will be checked
*  just to make sure the bitmap is valid.
*
* Instead, we will just read in another 4 bytes.
*

        bsr  wrbrbyte
        ldab #4
hdloop  jsr  rxbk
        bsr  wrbrbyte
        decb
        bne  hdloop

*
* Since we know the buffer has not flushed yet, we can read the data
* in it.
*
* Our intention is to read the length count, and divide it by 8.  If
*  a remainder exists, then we need to adjust the value by adding 1
*  to it.
*
* Dividing my 8 can be a difficult process since the data is not in
*  a proper form.  We must bit shift first 4 times.
*
* To avoid bit shifting, we can just do it once.  Instead of dividing
*  by 8, we can multiply by 2.  This will place the "decimal" point
*  in the proper location.
*

*
* Read the least significant length count (4 bits)
*

        ldaa brbufbeg+6

*
* Move its MSB onto the carry
*

        asla

*
* Now, shift the rest of the length cout (excluding the upper 4 bits)
```

```
*  only once to the left.
*
        ldd  brbufbeg+4
        rolb
        rola

*
* check the least significant length for a "mantissa"
*
        xgdy
*        brclr brbufbeg+6 $70 noincy

*
* If its present, increment the value
*
*        iny

*
* Store it for later use
*
noincy   sty  totalbyt

*
* Subtract 5 from it so we can use it as a counter to finish reading
*  the rest of the data.
*
        xgdy
        sec
        subd #5

*
* Main loop begins
*
* The value remaining to be received is stored in program memory.  This is
*  done because the remaining bytes will be required for a comparison later
*  on.
*
mnloop   std  byteslef

*
* Read a byte from the host computer
*
        jsr  rxbk

*
* Write it to the EEPROM
*
        bsr  wrbrbyte

*
* Decrement the counter by 1
*
```

```
        ldd  byteslef
        sec
        subd #1

*
* When it is zero, we are done
*

        bne  mnloop

*
* Flush buffer
*

        jsr  flushbuf

*
* Move back to address zero
*

        ldd  #0
        jsr  seteeadd

*
* Write the size at location zero
*

        ldd  totalbyt
        jsr  wordwrit

*
* Goto main
*

        jmp  main

lderror  jmp  errhand

*
* Write to EEPROM with burst buffer
*

wrbrbyte EQU  *

*
* Store the data in the buffer
*

        staa 0,y

*
* Increment the buffer pointer
*

        iny

*
* Check if we are at the end
*
```

```
        cpy  #brbufend
*
* If not, leave this routine
*
        beq  flushbuf
        rts
*
* Store the end of the buffer temporarily
*
flushbuf sty  flushlim
*
* Send an XOFF to tell the host to stop sending
*
        jsr  txxoff
*
* Set the buffer pointer to the beginning
*
        ldy  #brbufbeg
*
* Get the data from the buffer
*
wrbrloop ldaa 0,y
*
* Place it on the BUS
*
        staa portc,x
*
* Set /WE low on the EEPROM
*
        bclr portd,x $20
*
* Since the 68HC11 is realatively slow compared to the write timing requied
*  on the EEPROM, so we don't need any delays.
*
* Set /WE high on the EEPROM
*
        bset portd,x $20
*
* We now increment the EEPROM's address.
*
```

```
        jsr  inceeadd

*
* Increment the buffer pointer
*
        iny

*
* Compare the buffer pointer with flushlim
*

        cpy  flushlim

*
* If we are not done, send more
*

        bne  wrbrloop

*
* Performing a 10ms wait to complete the write to the EEPROM
*

        ldy #4290
decy    dey
        bne decy

*
* Reset buffer pointer
*

        ldy  #brbufbeg

*
* Tell the host it can send again
*

        jmp  txxon

*
* Tell the host computer that something is going on.  Output some "*"s
*

*         ldaa #$2a
*         jmp  tx
```

B.1.5.3 LDFPGA.ASM

The LDFPGA code performs two operations: load the FPGA with the contents of the upper external EEPROM memory and to move the captured data to the host.

```
*******************************************************************************
*
* File:  LDFPGA.ASM
```

```
*
* Dalsa Project FPGA loading and capture command level software for
*  the 68HC11 onboard microcontroller
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: March 28, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

*
* This program configures the FPGA with the data contained in the upper 16K
*  of the external EEPROM.  One this is done successfully, CAPTURE mode
*  is entered which moves the data from the FPGA to the host.
*

*
* BUFFALO Equates
*

prt0     EQU  $ae
prt1     EQU  $b0
prt2     EQU  $b2
prt3     EQU  $b4
prt4     EQU  $b6
prt5     EQU  $b8
prt6     EQU  $ba
prt7     EQU  $bc
prt8     EQU  $be

ldoffst  EQU  $ac
count    EQU  $a8
chrcnt   EQU  $a9

tmp1     EQU  $c0
tmp2     EQU  $c1
tmp3     EQU  $c2

shftreg  EQU  $96

hexbin   EQU  warmst+$09
dchek    EQU  $ffa6
upcase   EQU  $ffa0

*
* EEPROM routines
*

main     EQU  eeprom
errhand  EQU  main+3
seteeadd EQU  errhand+3
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
```

```
txxon    EQU  tx+3
txxoff   EQU  txxon+3
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3

BOE      EQU  $10
BWE      EQU  $20
BWSS     EQU  $08
BPROG    EQU  $04

*
* Set program origin to the start of free RAM
*

         ORG     rammid

ldfpga   EQU  *

*
* Use EEPROM function to initialize the ports
*

         jsr  initport

*
* Before the FPGA can be programmed, all the configuration memory must be
*  cleared.  This is done by setting /PROG on the FPGA low and waiting 300
*  microseconds.  To be safe we waited about 4 seconds.
*
* Set /WE high and /OE is high on the EEPROM
* Set /WS high and /PROG low on the FPGA
*

         ldaa #$38
         staa portd,x

*
* Delay of approximately 2 seconds at 8MHz
*

         ldy  #32768
clrdelay dey
         bne  clrdelay

*
* Tell the EEPROM to go into read mode and set the FPGA out of it's
*  configuration clearing mode.
*
* Set /WE high and /OE low on the EEPROM
* Set /WS high and /PROG high on the FPGA
*
```

```
        ldaa #$2c
        staa portd,x

*
* After all the configuration memory has been cleared we have to wait for
*  the FPGA to finish it's clearing cycle.  This happens when the /ERR or
*  /INIT line goes high.
*

initwait brclr porte,x $80 initwait

*
* Set EEPROM address to $0000
*
        ldd  #0
        jsr  seteeadd

*
* Read in the number of bytes to send to the FPGA
*

        jsr  wordread

*
* Increment the value by 1 to fixt the shift register program on the FPGA/
*

        xgdy
        iny

*
* Set the EEPROM in read mode
*

        bclr portd,x $10

*
* Jump into the loading routine.  This is done so that the DSP can be
*  properly booted.
*

        bra  firstskp

loadloop EQU  *

*
* Set PORT C to all inputs.
*

        clr  ddrc,x

*
* Increment the EEPROM's address.
*

        jsr  inceeadd

*
* Set EEPROM into read mode
```

```
*
* Set /WE high and /OE low on the EEPROM
* Set /WS high and /PROG high on the FPGA
*
        bclr portd,x $10
*
* If an error does not exist, skip the error output.
*
        brset porte,x $80 noperror
*
* Error, call global handler
*
        jmp  errhand
noperror EQU  *
*
* Read Data from EEPROM
*
firstskp ldab portc,x
*
* Turn off EEPROM
*
* Set /WE high and /OE high on the EEPROM
* Set /WS high and /PROG high on the FPGA
*
        bset portd,x $10
*
* Set PORT C to output
*
        bset ddrc,x  $ff
*
* Place data on BUS
*
        stab portc,x
*
* We now begin handshaking with the FPGA.  A low on /WS tells the FPGA that
*  data is ready on the BUS.
*
* Set /WE high and /OE high on the EEPROM
* Set /WS low and /PROG high on the FPGA
*
        bclr portd,x $8
*
* The following code is optional.  It simply outputs the values being sent to
```

```
*  the FPGA.
*

*         pshx
*         pshy
*         ldx  #regbas+portc
*         jsr  outlbsp
*         puly
*         pulx

*
* We now set /WS high again to tell the FPGA to use the data on the BUS.
*
* Set /WE high and /OE high on the EEPROM
* Set /WS high and /PROG high on the FPGA
*

         bset portd,x $8

*
* We now have to wait for the FPGA to fully process the data just passed.
*  The FPGA is ready for more data when /BUSY goes high.
*

*
* This routine is commented out since the FPGA guarentees this to be done
*  withing 60ns.  The microcontroller takes about 120ns per cycle.
*

busywait brclr porta,x $1 busywait

*
* Continue sending data until the Y has reached ZERO.
*

         dey
         bne  loadloop
*
* Set /WE high and /OE high on the EEPROM
* Set /WS high and /PROG high on the FPGA
*
* Start POP
*

         ldaa #$3c
         staa portd,x

*
* Assume the corrent number of bytes were sent, the FPGA's DONE line should
*  go high.  We will check this just to be sure.
*

donewait brclr porte,x $40 donewait

*
* The DONE line is high
*

*
```

```
* Command mode
*

CMRESET  EQU  $01
CMCAPON  EQU  $41
CMCAPOFF EQU  $42
CMFRESET EQU  $43
CMSEND   EQU  $44
CMGET    EQU  $45

command  equ  *

*
* Adjust port directions for serial communication to the FPGA
*

         ldaa #$fe
         staa ddrc,x

         ldaa #$00
         staa portc,x

*
* Output a "1" to indicate out level
*

command0 ldaa #$31
         jsr  tx

*
* Wait for a command
*

         jsr  rxbk

*
* Check for escape command
*

         cmpa #CMRESET
         bne  command1
         jmp  main

*
* For capture on/off, simple set static line
*

command1 cmpa #CMCAPON
         bne  command2
         bset portc,x $40

command2 cmpa #CMCAPOFF
         bne  command3
         bclr portc,x $40

*
* For FIFO reset, pulse reset line
*
```

```
command3 cmpa #CMFRESET
         bne  command4
         bset portc,x $80
         bclr portc,x $80

*
* Send a comand to the FPGA
*

command4 cmpa #CMSEND
         bne  command5
         jsr  rxbk
         tab
         jsr  rxbk
         bsr  putfpga
         bra  command0

*
* Move data from FIFO to the host while checking for a stop command
*

command5 cmpa #CMGET
         bne  command0

get0     jsr  rxnbk
         beq  get2
         cmpa #01
         beq  command0
get1     jsr  rxnbk
         beq  get1
         cmpa #01
         beq  command0
get2     bsr  getfpga
         bcs  get0
         jsr  tx
         bra  get0

*
* Get a byte from the FPGA
*

FPRCDA   EQU $1
FPRCCL   EQU $2

getfpga  EQU *

         bclr portd,x BWSS

getfpga0 brset scsr,x $20 getfpga2
         brclr porta,x $1 getfpga0

         ldy #8
getfpga1 ldaa portc,x
         lsra
         ror  ramlow
         bclr portc,x FPRCCL
         bset portc,x FPRCCL
         dey
         bne  getfpga1
```

```
        bset portd,x BWSS

        ldaa ramlow
        clc
        rts
getfpga2 sec
        rts

*
* Send a 10bit command to the FPGA
*

FPSDDA   EQU $4
FPSDCL   EQU $8
FPSDDN   EQU $10

putfpga  EQU *

        ldy #11
putfpga0 lsrd
        bcc putfpga1
        bset portc,x FPSDDA
        bra putfpga2

putfpga1 bclr portc,x FPSDDA

putfpga2 bclr portc,x FPSDCL
        bset portc,x FPSDCL
        dey
        bne putfpga0

        bclr portc,x FPSDDN
        bset portc,x FPSDDN
        rts
```

B.1.5.4 EEDELETE.ASM

The EEDELETE code deletes a file from the EEPROM filing system by removing the gap between the deleted file and the next file. The filing system does not support file replacement only additions, so the file must be deleted. This operation does not use any burst functions of the EEPROM, therefore it can take some time to remove a file at the beginning of the filing system.

```
*******************************************************************************
*
* File:  EEDELETE.ASM
*
* Dalsa Project EEPROM filing system file deleter for the onboard
*  68HC11 microcontroller
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
```

```
* Date Created: March 8, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

*
* BUFFALO Equates
*

prt0     EQU  $ae
prt1     EQU  $b0
prt2     EQU  $b2
prt3     EQU  $b4
prt4     EQU  $b6
prt5     EQU  $b8
prt6     EQU  $ba
prt7     EQU  $bc
prt8     EQU  $be

ldoffst  EQU  $ac
count    EQU  $a8
chrcnt   EQU  $a9

tmp1     EQU  $c0
tmp2     EQU  $c1
tmp3     EQU  $c2

shftreg  EQU  $96

hexbin   EQU  warmst+$09
dchek    EQU  $ffa6
upcase   EQU  $ffa0

*
* EEPROM routines
*

main     EQU  eeprom
errhand  EQU  main+3
seteeadd EQU  errhand+3
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
txxon    EQU  tx+3
txxoff   EQU  txxon+3
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3
```

```
*
* Some equates for this program
*

loadent  EQU  $c3
zeroram  EQU  $00
strtpnt  EQU  prt8
destpnt  EQU  prt7
movesiz  EQU  prt6
gapsiz   EQU  prt5
dirent   EQU  prt4

*
* Set program origin to the start of free RAM
*

         ORG  rammid

lddsp    EQU  *

         jsr  initport

*
* Get free data pointers
*

         jsr  strtload

*
* Store free start at zeroram+4
*

         ldd  zeroram
         std  zeroram+4
         ldd  zeroram+2
         std  zeroram+6

*
* Get deleting program pointers
*

         ldab loadent
         clra
         jsr  getptsz

*
* Check if its valid
*

         ldd  zeroram
         bne  entok
         jmp  sendok

entok    EQU  *

*
* Store start as dest pointer
*
```

```
        std  destpnt

*
* Find start point
*

        addd zeroram+2
        std  strtpnt

*
* Find the gap size
*

        sec
        subd destpnt
        std  gapsiz

*
* Find the total to be moved
*

        sec
        ldd  zeroram+4
        subd strtpnt
        std  movesiz

*
* Compinsate other pointers
*

        clra
        clrb

*
* Get pointer data
*

fixloop  std  dirent
        jsr  seteeadd

*
* Get start pointer
*

        jsr  wordread
        cpd  destpnt
        bcs  nextent

*
* If its after dest point, adjust the start
*

        sec
        subd gapsiz
        psha
        pshb
        ldd  dirent
        jsr  seteeadd
```

```
        pulb
        pula
        jsr  wordwrit

nextent ldd  dirent
        addd #4
        cpd  #16*4+4
        bne  fixloop

*
* Update free space
*

        ldd  #2
        jsr  seteeadd
        ldd  gapsiz
        addd zeroram+6
        jsr  wordwrit

        ldd  movesiz
        beq  lastprog

mvloop  ldd  strtpnt
        jsr  seteeadd
        jsr  byteread
        psha
        ldd  destpnt
        jsr  seteeadd
        pula
        jsr  bytewrit
        ldd  strtpnt
        addd #1
        std  strtpnt
        ldd  destpnt
        addd #1
        std  destpnt
        sec
        ldd  movesiz
        subd #1
        std  movesiz
        bne  mvloop

lastprog ldab loadent
        clra
        asld
        asld
        jsr  seteeadd
        clra
        clrb
        jsr  wordwrit
        jsr  wordwrit

*
* Delete OK, goto main
*

sendok  jsr  txxon
        jmp  main
```

B.1.5.5 LDEEPROC.ASM

The LDEEPROC code loads an arbitrary file into the EEPROM filing system. The file intended to be loaded contains information about the currently loaded FPGA bitstream in the upper 16K memory. Since the load time of the FPGA bitstream is significant, this much smaller file can be loaded to determine which bitstream is loaded to reduce overall wait time.

```
*******************************************************************************
*
* File:  LDEEPROC.ASM
*
* Dalsa Project Video processor definition file loading software for
*  the 68HC11 onboard microcontroller
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: September 23, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

*
* BUFFALO Equates
*

prt0     EQU  $ae
prt1     EQU  $b0
prt2     EQU  $b2
prt3     EQU  $b4
prt4     EQU  $b6
prt5     EQU  $b8
prt6     EQU  $ba
prt7     EQU  $bc
prt8     EQU  $be

ldoffst  EQU  $ac
count    EQU  $a8
chrcnt   EQU  $a9

tmp1     EQU  $c0
tmp2     EQU  $c1
tmp3     EQU  $c2

shftreg  EQU  $96

hexbin   EQU  warmst+$09
dchek    EQU  $ffa6
upcase   EQU  $ffa0

*
* EEPROM routines
```

```
*

main     EQU  eeprom
errhand  EQU  main+3
seteeadd EQU  errhand+3
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
txxon    EQU  tx+3
txxoff   EQU  txxon+3
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3

*
* Some extra equates for this program
*

prgstrt  EQU  prt6
written  EQU  prt7
secsize  EQU  prt8

*
* Set program origin to the start of free RAM
*

         ORG  rammid

lds19    EQU  *

*
* Call EEPROM to initialize ports
*

         jsr  initport

*
* Create file
*

         jsr  strtload

*
* Place code at the beginning of the file to jump to error handler
*  incase this is accidently executed
*

         ldaa #$7e
         jsr  bytewrit
         ldd  #errhand
```

```
        jsr  wordwrit

*
* Get and save definition size
*

        jsr  rxbkflow
        psha
        jsr  rxbkflow
        tab
        pula
        std  secsize
        jsr  wordwrit
        ldy  secsize

*
* Get N bytes and save them
*

getmore  jsr  rxbkflow
        jsr  bytewrit
        dey
        bne  getmore

*
* Close file, leave
*

        jsr  finiload

        jsr  txxon
        jmp  main
```

B.1.5.6 GETPROC.ASM

The GETPROC code retrieves the data stored into the EEPROM filing system by the EELDPROC code.

```
*******************************************************************************
*
* File:  GETPROC.ASM
*
* Dalsa Project Video processor definition file retrieving software for
*  the 68HC11 onboard microcontroller
* NOT FOR PRODUCTION
* Copyright University of Windsor 1996-1999
* Date Created: September 23, 1996
*
*******************************************************************************
*
*#INCLUDE 'hc11reg.equ'
*

*
```

```
* BUFFALO Equates
*

prt0     EQU  $ae
prt1     EQU  $b0
prt2     EQU  $b2
prt3     EQU  $b4
prt4     EQU  $b6
prt5     EQU  $b8
prt6     EQU  $ba
prt7     EQU  $bc
prt8     EQU  $be

ldoffst  EQU  $ac
count    EQU  $a8
chrcnt   EQU  $a9

tmp1     EQU  $c0
tmp2     EQU  $c1
tmp3     EQU  $c2

shftreg  EQU  $96

hexbin   EQU  warmst+$09
dchek    EQU  $ffa6
upcase   EQU  $ffa0

*
* EEPROM routines
*

main     EQU  eeprom
errhand  EQU  main+3
seteeadd EQU  errhand+3
geteeadd EQU  seteeadd+3
inceeadd EQU  geteeadd+3
rxnbk    EQU  inceeadd+3
rxbk     EQU  rxnbk+3
rxbkflow EQU  rxbk+3
tx       EQU  rxbkflow+3
txxon    EQU  tx+3
txxoff   EQU  txxon+3
byteread EQU  txxoff+3
wordread EQU  byteread+3
loadprg  EQU  wordread+3
getptsz  EQU  loadprg+3
bytewrit EQU  getptsz+3
wordwrit EQU  bytewrit+3
strtload EQU  wordwrit+3
finiload EQU  strtload+3
initport EQU  finiload+3

*
* Some extra equates for this program
*

prgstrt  EQU  prt6
written  EQU  prt7
secsize  EQU  prt8
```

```
*
* Set program origin to the start of free RAM
*

zeroram  EQU  ramlow

         ORG  rammid

lds19    EQU  *

*
* Call EEPROM to initialize ports
*

         jsr  initport

         clra
         staa tmp1
         staa shftreg+1

*
* Get directory entry
*

         jsr  rxbk
         jsr  hexbin

         clra
         ldab shftreg+1

         jsr  getptsz
         ldd  zeroram
         beq  allzero

         jsr  seteeadd

*
* Skip safety code
*

         jsr  inceeadd
         jsr  inceeadd
         jsr  inceeadd

*
* Get size and output it
*

         jsr  wordread
allzero  std  zeroram+2
         jsr  tx
         tba
         jsr  tx
         ldy  zeroram+2
         beq  leave

*
* Output contents
```

```
*

getmore  jsr  byteread
         jsr  tx
         dey
         bne  getmore

*
* Go back to main
*

leave    jmp  main
```

B.2 FPGA Hardware Description Code

B.2.1 Synopsys Design Compiler Procedure and Scripts

Before Design Compiler can be executed, a setup file (.synopsys_dc.setup) must be located in the current directory which contains the information of the target technology. Design Compiler can then be launched with either "dc_shell" (text based) or "design_analyzer" (GUI based) assuming the correct Synopsys program paths are set. Once Design Compiler is loaded, a command to load the video processor should be executed: "read -f vhdl <videoprocessor>.vhd". After the code is successfully read in, the automated script can be executed with "include makefpga.scr". Creation of the logic for the whole FPGA system can take up to 20 minutes. Once it is complete, the Xilinx tools are needed for the final implementation steps.

B.2.1.1 .synopsys_dc.setup

```
/* Set information for schematic sheets */
designer = "Roberto Muscedere";
company = "VLSI Research Group";

/* Set synthetic library path, for synopsys libs and xact libs */
search_path = {. /usr/local/vlsi/tools/synopsys/libraries/syn \
/usr/local/vlsi/tools/xact/synopsys/libraries/syn}
/* Set target hardware to 4005-5, include primatives and specific libraries */
link_library = {"*" xprim_4005-5.db xprim_4000-5.db xgen_4000.db xdc_4000-5.db \
xio_4000-5.db xfpga_4000-5.db}
target_library = {xprim_4005-5.db xprim_4000-5.db xgen_4000.db xdc_4000-5.db \
xio_4000-5.db xfpga_4000-5.db}
/* Add symbol library (for gui interface) */
symbol_library = xc4000.sdb
/* Add XBLOX synthetic libraries to improve performance */
synthetic_library = {xblox_4000.sldb standard.sldb}
define_design_lib xblox_4000 -path \
        /usr/local/vlsi/tools/xact/synopsys/libraries/dw/lib/fpga/xc4000
/* Setup outputs for Xilinx XNF file format */
compile_fix_multiple_port_nets = true
bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"
bus_inference_style = "%s<%d>"
edifout_netlist_only = true
edifout_power_and_ground_representation = cell
edifout_write_properties_list = "instance_number port_location part"
xlnx_hier_blknm = 1
xnfout_library_version = "2.0.0"
```

B.2.1.2 makefpga.scr

```
/* resource allocation may need to be turned off */
```

```
/*   hlo_resource_allocation = none */

/* make sure to read a video processor prior to running this script */
/* eg: read -f vhdl process.vhd */

/* Read in external fifo code */
    read -f vhdl extfifo.vhd
/* Read in internal fifo code */
    read -f vhdl intfifo.vhd
/* Set special cells to don't touch so synopsys won't optimize them out */
    set_dont_touch xdram*

/* Read in host communication code */
    read -f vhdl host.vhd
/* Read in system linking code */
    read -f vhdl link.vhd

/* Set out target design (from linking code) */
    current_design larch1

/* Disable all clock pins, we only have 1 we can use in the FPGA */
    set_pad_type -no_clock "*"
/* Set the proper clock pin */
    set_pad_type -clock {CLK}
/* Set the RAM DATA lines so they don't float */
    set_pad_type -pulldown {RAMDATA}
/* Set the slew rate */
    set_pad_type -slewrate HIGH all_outputs()

/* Set all ports to be pads on the final design */
    set_port_is_pad "*"

/* Set all pin assignments from an external file */
    include fpgapins.scr

/* Read pad information from target library */
    insert_pads

/* Set clock speed to just below 15 MHz */
    create_clock CLK -period  62

/* Compile the design to the target technology */
    compile -map_effort high

/* Report some approximate information to the user */
    report_fpga
    report_timing

/* Convert the whole design into gates (synopsys cannot export */
/*  4000 series bitmaps directly, we need the Xilinx tools */
    replace_fpga

/* Set default values for the flipflops */
    current_design ext_fifo_control
    include fifo_init.scr
    current_design larch1

/* Set the target FPGA type */
    set_attribute larch1 "part" -type string "4005epc84-2"
```

```
/* Set some XNF file properties */
   set_attribute find(design,"*") "xnfout_use_blknames" -type boolean FALSE

/* Write out Xilinx XNF file */
   write -format xnf -hierarchy -output fpga.sxnf
```

B.2.1.3 fpgapins.scr

```
/* Static RAM (FIFO) Data lines */

set_attribute {"RAMDATA<0>"} "pad_location" -type string "P27"
set_attribute {"RAMDATA<1>"} "pad_location" -type string "P26"
set_attribute {"RAMDATA<2>"} "pad_location" -type string "P25"
set_attribute {"RAMDATA<3>"} "pad_location" -type string "P24"
set_attribute {"RAMDATA<4>"} "pad_location" -type string "P23"
set_attribute {"RAMDATA<5>"} "pad_location" -type string "P20"
set_attribute {"RAMDATA<6>"} "pad_location" -type string "P19"
set_attribute {"RAMDATA<7>"} "pad_location" -type string "P18"

/* Static RAM (FIFO) Address lines */

set_attribute {"RAMADDR<0>"} "pad_location" -type string "P47"
set_attribute {"RAMADDR<1>"} "pad_location" -type string "P46"
set_attribute {"RAMADDR<2>"} "pad_location" -type string "P45"
set_attribute {"RAMADDR<3>"} "pad_location" -type string "P44"
set_attribute {"RAMADDR<4>"} "pad_location" -type string "P40"
set_attribute {"RAMADDR<5>"} "pad_location" -type string "P39"
set_attribute {"RAMADDR<6>"} "pad_location" -type string "P38"
set_attribute {"RAMADDR<7>"} "pad_location" -type string "P28"
set_attribute {"RAMADDR<8>"} "pad_location" -type string "P48"
set_attribute {"RAMADDR<9>"} "pad_location" -type string "P49"
set_attribute {"RAMADDR<10>"} "pad_location" -type string "P50"
set_attribute {"RAMADDR<11>"} "pad_location" -type string "P62"
set_attribute {"RAMADDR<12>"} "pad_location" -type string "P68"
set_attribute {"RAMADDR<13>"} "pad_location" -type string "P66"
set_attribute {"RAMADDR<14>"} "pad_location" -type string "P80"
set_attribute {"RAMADDR<15>"} "pad_location" -type string "P81"
set_attribute {"RAMADDR<16>"} "pad_location" -type string "P82"

/* Static RAM (FIFO) Control lines */

set_attribute {"RAMOE"} "pad_location" -type string "P83"
set_attribute {"RAMWE"} "pad_location" -type string "P84"

/* Camera signals */

set_attribute {"CLK"} "pad_location" -type string "P78" /* 29,51,"P78" */
set_attribute {"LINEVALID"} "pad_location" -type string "P10"
set_attribute {"PIXELVALID"} "pad_location" -type string "P51"
set_attribute {"CAMERADATA<0>"} "pad_location" -type string "P3"
set_attribute {"CAMERADATA<1>"} "pad_location" -type string "P4"
set_attribute {"CAMERADATA<2>"} "pad_location" -type string "P5"
set_attribute {"CAMERADATA<3>"} "pad_location" -type string "P6"
set_attribute {"CAMERADATA<4>"} "pad_location" -type string "P7"
set_attribute {"CAMERADATA<5>"} "pad_location" -type string "P8"
set_attribute {"CAMERADATA<6>"} "pad_location" -type string "P9"
set_attribute {"CAMERADATA<7>"} "pad_location" -type string "P14"
```

```
/* Microcontroller signals */

set_attribute {"READY"} "pad_location" -type string "P70"
set_attribute {"REQUEST"} "pad_location" -type string "P77"
set_attribute {"EMPTY"} "pad_location" -type string "P36"
set_attribute {"SENDDATA"} "pad_location" -type string "P56"
set_attribute {"SENDCLOCK"} "pad_location" -type string "P58"
set_attribute {"RECEIVEDATA"} "pad_location" -type string "P59"
set_attribute {"RECEIVECLOCK"} "pad_location" -type string "P61"
set_attribute {"RECEIVEDONE"} "pad_location" -type string "P65"
/* set_attribute {"WRITEDATA<5>"} "pad_location" -type string "P67" */
set_attribute {"CAPTUREON"} "pad_location" -type string "P69"
set_attribute {"FIFORESET"} "pad_location" -type string "P71"
```

B.2.1.4 fifo_init.scr

```
set_attribute "C1RAMOE_reg" xnf_init -type string "S"
set_attribute "C1RAMWE_reg" xnf_init -type string "S"
set_attribute "C2RAMOE_reg" xnf_init -type string "S"
set_attribute "C2RAMWE_reg" xnf_init -type string "S"
set_attribute "EMPTY_reg" xnf_init -type string "S"
set_attribute "FULL_reg" xnf_init -type string "R"
set_attribute "WPOINT_reg<0>" xnf_init -type string "R"
set_attribute "RPOINT_reg<0>" xnf_init -type string "R"
```

B.2.2 Xilinx Scripts

The Xilinx optimizer and automatic place and route are all executed from a simple command: "xmake". Before this can be done, a minor change must be made to the source file. Design Compiler insists on making the system clock with a secondary clock buffer, however, in the physical design the system clock is connected to a primary clock buffer. A series of command is sent to Unix "ed" program which will change the buffer type. If this is not done, the optimizer will error out. After this whole process, the bitstream "FPGA.bit" is created.

B.2.2.1 makefpga

```
#!/bin/sh

# Change BUGGS to BUFGP in fpga.sxnf

ed fpga.sxnf << END
g/BUFG/s/BUFGS/BUFGP/g
w
q
END

# Run Xilinx make file
xmake fpga
```

B.2.3 Main VHDL Code

B.2.3.1 link.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define system ports
entity larch1 is
    port(CLK: in STD_LOGIC;
    CAMERADATA: in UNSIGNED(7 downto 0);
    RAMADDR: out STD_LOGIC_VECTOR(16 downto 0);
    RAMDATA: inout STD_LOGIC_VECTOR(7 downto 0);
    RAMOE: out STD_LOGIC;
    RAMWE: out STD_LOGIC;
    PIXELVALID: in STD_LOGIC;
    LINEVALID: in STD_LOGIC;

    CAPTUREON: in STD_LOGIC;
    FIFORESET: in STD_LOGIC;

    RECEIVEDATA: in STD_LOGIC;
    RECEIVECLOCK: in STD_LOGIC;
    RECEIVEDONE: in STD_LOGIC;
    SENDDATA: out STD_LOGIC;
    SENDCLOCK: in STD_LOGIC;

    REQUEST: in STD_LOGIC;
    READY: out STD_LOGIC );
end larch1;

-- define system behaviour
architecture behaviour of larch1 is

-- descirbe subsystems
component ext_fifo_control
    port(WDATA: in STD_LOGIC_VECTOR(7 downto 0);
    RDATA: out STD_LOGIC_VECTOR(7 downto 0);
    RAMADDR: out STD_LOGIC_VECTOR(16 downto 0);
    RAMDATA: inout STD_LOGIC_VECTOR(7 downto 0);
    RAMOE: out STD_LOGIC;
    RAMWE: out STD_LOGIC;
    FULL: buffer STD_LOGIC;
    EMPTY: buffer STD_LOGIC;
    CLK: in STD_LOGIC;
    WRITE: in STD_LOGIC;
    BUSY: out STD_LOGIC;
    RESET: in STD_LOGIC;
    READ: in STD_LOGIC;
    READDONE: out STD_LOGIC );
end component;

component camera_interface
    port(CLK: in STD_LOGIC;
    CAMERADATA: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(15 downto 0);
```

```
    OUTDATA16: out STD_LOGIC;
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;

    COMMANDDATA: in STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: in STD_LOGIC;

    CAPTUREON: in STD_LOGIC;

    PUSH: out STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end component;

component host_interface
    port(CLK: in STD_LOGIC;

    COMMANDDATAOUT: buffer STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: out STD_LOGIC;
    POP: out STD_LOGIC;

    DATAIN: in STD_LOGIC_VECTOR(7 downto 0);
    OKTOPOP: in STD_LOGIC;
    POPDONE: in STD_LOGIC;

    RECEIVEDATA: in STD_LOGIC;
    RECEIVECLOCK: in STD_LOGIC;
    RECEIVEDONE: in STD_LOGIC;
    SENDDATA: out STD_LOGIC;
    SENDCLOCK: in STD_LOGIC;

    REQUEST: in STD_LOGIC;
    READY: out STD_LOGIC );
end component;

component int_fifo_control
    port(CLK: in STD_LOGIC;
    DATAIN: in STD_LOGIC_VECTOR(15 downto 0);
    DATAOUT: out STD_LOGIC_VECTOR(7 downto 0);
    WRITEIN: in STD_LOGIC;
    WRITE16: in STD_LOGIC;
    WRITEOUT: out STD_LOGIC;
    BUSY: in STD_LOGIC;
    RESET: in STD_LOGIC );
end component;

-- define subsystem connecting signals
signal ICOMMDATA: STD_LOGIC_VECTOR(10 downto 0);
signal IIWDATA: STD_LOGIC_VECTOR(15 downto 0);
signal IEWDATA: STD_LOGIC_VECTOR(7 downto 0);
signal IRDATA: STD_LOGIC_VECTOR(7 downto 0);
signal IMODE16: STD_LOGIC;
signal IIPUSH: STD_LOGIC;
signal IEPUSH: STD_LOGIC;
signal IBUSY: STD_LOGIC;
signal IPOP: STD_LOGIC;
signal IPOPDONE: STD_LOGIC;
signal IEMPTY: STD_LOGIC;
signal IFULL: STD_LOGIC;
signal ICOMMDATAVALID: STD_LOGIC;
```

```
signal ICAPTUREON: STD_LOGIC;
signal IFIFORESET: STD_LOGIC;

begin

-- stabalize a few signals
    stabalize_signals: process begin
    wait until CLK'event and CLK='0';

    ICAPTUREON <= CAPTUREON;
    IFIFORESET <= FIFORESET;

    end process;

-- instantiate sub-systems with connectivity
xcamera: camera_interface port map (OUTDATA => IIWDATA, CLK => CLK,
    CAMERADATA => CAMERADATA, PUSH => IIPUSH, OKTOPUSH => IFULL,
    LVAL => LINEVALID, PVAL => PIXELVALID, CAPTUREON => ICAPTUREON,
    OUTDATA16 => IMODE16,
    COMMANDDATA => ICOMMDATA, COMMANDDATAVALID => ICOMMDATAVALID);

xififo: int_fifo_control port map (DATAIN => IIWDATA, DATAOUT => IEWDATA,
    WRITEIN => IIPUSH, WRITEOUT => IEPUSH, BUSY => IBUSY,
    WRITE16 => IMODE16,
    RESET => IFIFORESET, CLK => CLK);

xefifo: ext_fifo_control port map (WDATA => IEWDATA, RDATA => IRDATA,
    RAMADDR => RAMADDR, RAMOE => RAMOE, RAMWE => RAMWE, CLK => CLK,
    WRITE => IEPUSH, READ => IPOP, READDONE => IPOPDONE,
    EMPTY => IEMPTY, FULL => IFULL, RAMDATA => RAMDATA,
    BUSY => IBUSY, RESET => IFIFORESET);

xhost: host_interface port map(CLK => CLK, REQUEST => REQUEST,
    READY => READY, POP => IPOP, POPDONE => IPOPDONE, OKTOPOP => IEMPTY,
    DATAIN => IRDATA, RECEIVEDONE => RECEIVEDONE,
    RECEIVEDATA => RECEIVEDATA, RECEIVECLOCK => RECEIVECLOCK,
    SENDDATA => SENDDATA, SENDCLOCK => SENDCLOCK,
    COMMANDDATAOUT => ICOMMDATA, COMMANDDATAVALID => ICOMMDATAVALID);

end behaviour;

-- Following code is necessary for simulation with vhdldbx

-- configuration config_test_efc of test_efc is
--  for behaviour
--      for xfifo: ext_fifo_control use entity work.ext_fifo_control(behaviour);
--      end for;
--      for xtest: efc_test use entity work.efc_test(behaviour);
--      end for;
--      for xhost: host_interface use entity work.host_interface(behaviour);
--      end for;
--      for xport: ramport_control use entity work.ramport_control(behaviour);
--      end for;
--  end for;
-- end config_test_efc;
```

B.2.3.2 intfifo.vhd

```
-- Include necessary libaries
```

```
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define sub-system ports
entity int_fifo_control is
    port(CLK: in STD_LOGIC;
    DATAIN: in STD_LOGIC_VECTOR(15 downto 0);
    DATAOUT: out STD_LOGIC_VECTOR(7 downto 0);
    WRITEIN: in STD_LOGIC;
    WRITE16: in STD_LOGIC;
    WRITEOUT: out STD_LOGIC;
    BUSY: in STD_LOGIC;
    RESET: in STD_LOGIC );
end int_fifo_control;

architecture behaviour of int_fifo_control is

-- define Xilinx DP 16x2 RAM component (falling edge)
component RAMD
    port(A0: in STD_LOGIC;
    A1: in STD_LOGIC;
    A2: in STD_LOGIC;
    A3: in STD_LOGIC;
    DPRA0: in STD_LOGIC;
    DPRA1: in STD_LOGIC;
    DPRA2: in STD_LOGIC;
    DPRA3: in STD_LOGIC;
    WE: in STD_LOGIC;
    WCLK: in STD_LOGIC;
    D: in STD_LOGIC;
--  SPO: out STD_LOGIC;
    DPO: out STD_LOGIC );
end component;

-- Set up a variable for the clock edges
constant CYCLE1: STD_LOGIC := '0';
constant CYCLE2: STD_LOGIC := '1';
signal WPS: STD_LOGIC_VECTOR(3 downto 0);
signal RPS: STD_LOGIC_VECTOR(3 downto 0);
signal WP: STD_LOGIC_VECTOR(3 downto 0);
signal RP: STD_LOGIC_VECTOR(3 downto 0);
signal TAG: STD_LOGIC;
signal WE: STD_LOGIC;
signal DATAL: STD_LOGIC_VECTOR(7 downto 0);
signal DATAH: STD_LOGIC_VECTOR(7 downto 0);
signal DATALO: STD_LOGIC_VECTOR(7 downto 0);
signal DATAHO: STD_LOGIC_VECTOR(7 downto 0);
signal INDEX: STD_LOGIC;
signal INDEX2: STD_LOGIC;
signal INCRP: STD_LOGIC;
signal IRESET: STD_LOGIC;
begin

-- stabalize reset line
-- cycle 1, process 3
    int_fifo_cycle1_3: process begin
    wait until CLK'event and CLK=CYCLE1;
```

```
    IRESET <= RESET;

    end process;

-- set write line immediately
-- cycle 1, process 1
    int_fifo_cycle1_1: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (WRITEIN='1') then
        WE <= '1';
    else
        WE <= '0';
    end if;

    end process;

-- set write pointer, reset it if necessary
-- cycle 2, process 1
    int_fifo_cycle2_1: process(IRESET,CLK)
    begin

    if (IRESET='1') then
        WP <= "0000";
    elsif (CLK'event and CLK=CYCLE2) then
        if (WE='1') then
            WP <= WP + 1;
        end if;
    end if;

    end process;

-- move data from internal to external
-- cycle 1, process 2
    int_fifo_cycle1_2: process begin
    wait until CLK'event and CLK=CYCLE1;

    INDEX2 <= INDEX;
    if (BUSY='0' and not (WP=RP)) then
        WRITEOUT <= '1';
        if (INDEX='0') then
            if (TAG='0') then
                INDEX <= '0';
                INCRP <= '1';
            else
                INDEX <= '1';
                INCRP <= '0';
            end if;
        else
            INDEX <= '0';
            INCRP <= '1';
        end if;
    else
        WRITEOUT <= '0';
        INDEX <= '0';
        INCRP <= '0';
    end if;
```

```
    end process;

-- update read pointer, reset if necessary
-- cycle 2, process 2
    int_fifo_cycle2_2: process(IRESET,CLK)
    begin

    if (IRESET='1') then
        RP <= "0000";
    elsif (CLK'event and CLK=CYCLE2) then
        if (INCRP='1') then
            RP <= RP + 1;
        end if;
    end if;

    end process;

-- use tri-state instead of multiplexers to CLB save space
    int_fifo_unclocked_1: process (INDEX2,DATAL,DATAH)
    begin

    if (INDEX2='0') then
        DATALO <= DATAL;
        DATAHO <= (others => 'Z');
    else
        DATALO <= (others => 'Z');
        DATAHO <= DATAH;
    end if;

    end process;

-- connect clocked signals to RAM components
    int_fifo_unclocked_2: process (WP,RP) begin

    WPS <= WP;
    RPS <= RP;

    end process;

    int_fifo_unclocked_3: process (DATALO) begin

    DATAOUT <= DATALO;

    end process;

    int_fifo_unclocked_4: process (DATAHO) begin

    DATAOUT <= DATAHO;

    end process;

-- instantiate 17 DP RAM blocks
    xdram00: RAMD port map ( D => DATAIN(0), DPO => DATAL(0),
        A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
        DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
        WCLK => CLK, WE => WE );

    xdram01: RAMD port map ( D => DATAIN(1), DPO => DATAL(1),
        A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
```

```
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram02: RAMD port map ( D => DATAIN(2), DPO => DATAL(2),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram03: RAMD port map ( D => DATAIN(3), DPO => DATAL(3),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram04: RAMD port map ( D => DATAIN(4), DPO => DATAL(4),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram05: RAMD port map ( D => DATAIN(5), DPO => DATAL(5),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram06: RAMD port map ( D => DATAIN(6), DPO => DATAL(6),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram07: RAMD port map ( D => DATAIN(7), DPO => DATAL(7),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram08: RAMD port map ( D => DATAIN(8), DPO => DATAH(0),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram09: RAMD port map ( D => DATAIN(9), DPO => DATAH(1),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram10: RAMD port map ( D => DATAIN(10), DPO => DATAH(2),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram11: RAMD port map ( D => DATAIN(11), DPO => DATAH(3),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram12: RAMD port map ( D => DATAIN(12), DPO => DATAH(4),
    A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
    DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
    WCLK => CLK, WE => WE );

xdram13: RAMD port map ( D => DATAIN(13), DPO => DATAH(5),
```

```
        A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
        DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
        WCLK => CLK, WE => WE );

    xdram14: RAMD port map ( D => DATAIN(14), DPO => DATAH(6),
        A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
        DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
        WCLK => CLK, WE => WE );

    xdram15: RAMD port map ( D => DATAIN(15), DPO => DATAH(7),
        A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
        DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
        WCLK => CLK, WE => WE );

    xdramtag: RAMD port map ( D => WRITE16, DPO => TAG,
        A0 => WPS(0), A1 => WPS(1), A2 => WPS(2), A3 => WPS(3),
        DPRA0 => RPS(0), DPRA1 => RPS(1), DPRA2 => RPS(2), DPRA3 => RPS(3),
        WCLK => CLK, WE => WE );

end behaviour;
```

B.2.3.3 extfifo.vhd

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define sub-system ports
entity ext_fifo_control is
    port(WDATA: in STD_LOGIC_VECTOR(7 downto 0);
    RDATA: out STD_LOGIC_VECTOR(7 downto 0);
    RAMADDR: out STD_LOGIC_VECTOR(16 downto 0);
    RAMDATA: inout STD_LOGIC_VECTOR(7 downto 0);
    RAMOE: out STD_LOGIC;
    RAMWE: out STD_LOGIC;
    FULL: buffer STD_LOGIC;
    EMPTY: buffer STD_LOGIC;
    CLK: in STD_LOGIC;
    WRITE: in STD_LOGIC;
    BUSY: out STD_LOGIC;
    RESET: in STD_LOGIC;
    READ: in STD_LOGIC;
    READDONE: out STD_LOGIC );
end ext_fifo_control;

architecture behaviour of ext_fifo_control is
constant CYCLE1: STD_LOGIC := '1';
constant CYCLE2: STD_LOGIC := '0';
signal WPOINT: STD_LOGIC_VECTOR(16 downto 0) := "00000000000000000";
signal RPOINT: STD_LOGIC_VECTOR(16 downto 0) := "00000000000000000";
signal FSIZE: STD_LOGIC_VECTOR(16 downto 0) := "00000000000000000";
signal C1RAMOE: STD_LOGIC := '1';
signal C2RAMOE: STD_LOGIC := '1';
signal C1RAMWE: STD_LOGIC := '1';
signal C2RAMWE: STD_LOGIC := '1';
signal RAMWRITE: STD_LOGIC_VECTOR(7 downto 0);
signal DATALINE: STD_LOGIC_VECTOR(7 downto 0);
```

```
signal READDATA: STD_LOGIC_VECTOR(7 downto 0);
-- signal TESTCOUNT: STD_LOGIC_VECTOR(7 downto 0);
signal TRICONTROL: STD_LOGIC := '0';
signal RAMSTATE: STD_LOGIC := '0';
signal STATE: STD_LOGIC_VECTOR(1 downto 0);
signal IRESET: STD_LOGIC;
begin

-- note the use of many processes
--  this is to make simulation easier

-- cycle 1, process 1
    fifo_control_cycle1_1: process begin
    wait until CLK'event and CLK=CYCLE1;

    RAMWRITE <= WDATA;

    end process;

-- cycle 1, process 4
    fifo_control_cycle1_4: process(IRESET,CLK)
    begin

-- during reset, set all signals
    if (IRESET='1') then

        READDONE <= '0';
        C2RAMOE <= '1';
        C2RAMWE <= '1';
        TRICONTROL <= '1';
        BUSY <= '1';
        RAMSTATE <= '0';
        WPOINT <= (others => '0');
        RPOINT <= (others => '0');
        FSIZE <= (others => '0');
        FULL <= '0';
        EMPTY <= '1';
        STATE <= "00";

    elsif( CLK'event and CLK=CYCLE1 ) then

-- use signal bus to indicte our current write/read state
        case STATE is
            when "00" =>

                if (WRITE='1' and FULL='0') then

-- write, fifo not full
                    READDONE <= '0';
                    C2RAMOE <= '1';
                    C2RAMWE <= '0';
                    TRICONTROL <= '1';
                    BUSY <= '0';
                    RAMSTATE <= '0';
                    RAMADDR <= WPOINT;
                    WPOINT <= WPOINT + 1;
                    FSIZE <= FSIZE + 1;
-- update full state
                    if (FSIZE="1111111111110000") then
```

```
                    FULL <= '1';
                else
                    FULL <= '0';
                end if;
                EMPTY <= '0';
                STATE <= "00";

            elsif (READ='1' and EMPTY='0') then

-- read, fifo not empty (stage 1)
                READDONE <= '0';
                C2RAMOE <= '1';
                C2RAMWE <= '1';
                TRICONTROL <= '0';
                BUSY <= '1';
                RAMSTATE <= '0';
                RAMADDR <= RPOINT;
                STATE <= "01";

            else

-- idle, do nothing
                READDONE <= '0';
                C2RAMOE <= '1';
                C2RAMWE <= '1';
                TRICONTROL <= '1';
                BUSY <= '0';
                RAMSTATE <= '0';
                RAMADDR <= WPOINT;
                STATE <= "00";

            end if;

        when "01" =>

-- read (stage 2)
            READDONE <= '0';
            C2RAMOE <= '0';
            C2RAMWE <= '1';
            TRICONTROL <= '0';
            BUSY <= '1';
            RAMSTATE <= '1';
            RAMADDR <= RPOINT;
            STATE <= "10";

        when "10" =>

-- read (stage 3)
            READDONE <= '1';
            C2RAMOE <= '1';
            C2RAMWE <= '1';
            TRICONTROL <= '0';
            BUSY <= '1';
            RAMSTATE <= '0';
            RPOINT <= RPOINT + 1;

            FSIZE <= FSIZE - 1;
            RAMADDR <= RPOINT;
-- update empty flag
```

```
            if (FSIZE="0000000000000001") then
                EMPTY <= '1';
            else
                EMPTY <= '0';
            end if;
            FULL <= '0';
            RDATA <= READDATA;
            STATE <= "00";

        when others =>

-- unknown states handle to prevent instatiation of latches
            READDONE <= '0';
            C2RAMOE <= '1';
            C2RAMWE <= '1';
            TRICONTROL <= '1';
            BUSY <= '0';
            RAMSTATE <= '0';
            RAMADDR <= WPOINT;
            STATE <= "00";

    end case;

end if;

end process;

-- state SRAM control signals for one phase
-- cycle 2, process 1
fifo_control_cycle2_1: process begin
wait until CLK'event and CLK=CYCLE2;

C1RAMWE <= '1';
if (RAMSTATE='1') then
    C1RAMOE <= '0';
else
    C1RAMOE <= '1';
end if;

end process;

-- stabalize reset signal
-- cycle 2, process 2
fifo_control_cycle2_2: process begin
wait until CLK'event and CLK=CYCLE2;

IRESET <= RESET;

end process;

-- Set SRAM control lines for other clock phase
-- always process 1 (no clocking)
efc_always_1: process (CLK,C1RAMOE,C1RAMWE,C2RAMOE,C2RAMWE) begin

-- set sram controls based on current cycle
case CLK is
    when CYCLE1 =>
        RAMOE <= C1RAMOE;
        RAMWE <= C1RAMWE;
```

```
        when CYCLE2 =>
            RAMOE <= C2RAMOE;
            RAMWE <= C2RAMWE;
        when others => NULL;
    end case;
    end process;

-- always process 2 (no clocking)
    efc_always_2: process (TRICONTROL,RAMWRITE) begin

-- set 8 tristates from one control line
-- used to make tristate controls more implicit
    if (TRICONTROL='1') then
        DATALINE <= RAMWRITE;
    else
        DATALINE <= "ZZZZZZZZ";
    end if;
    end process;

-- always process 5 (no clocking)
    efc_always_5: process (DATALINE,RAMWRITE) begin

-- used to make tristate controls more implicit
    RAMDATA <= DATALINE;
    end process;

-- always process 6 (no clocking)
    efc_always_6: process (RAMDATA) begin

-- used to make tristate controls more implicit
    READDATA <= RAMDATA;
    end process;

end behaviour;
```

B.2.3.4 host.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity host_interface is
    port(CLK: in STD_LOGIC;

    COMMANDDATAOUT: buffer STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: out STD_LOGIC;
    POP: out STD_LOGIC;

    DATAIN: in STD_LOGIC_VECTOR(7 downto 0);
    OKTOPOP: in STD_LOGIC;
    POPDONE: in STD_LOGIC;

    RECEIVEDATA: in STD_LOGIC;
    RECEIVECLOCK: in STD_LOGIC;
    RECEIVEDONE: in STD_LOGIC;
    SENDDATA: out STD_LOGIC;
    SENDCLOCK: in STD_LOGIC;
```

```
    REQUEST: in STD_LOGIC;
    READY: out STD_LOGIC );
end host_interface;

architecture behaviour of host_interface is
constant CYCLE1: STD_LOGIC := '0';
constant CYCLE2: STD_LOGIC := '1';
constant READSTART: STD_LOGIC := '0';
constant READEND: STD_LOGIC := '1';
signal READSIG: STD_LOGIC;
signal READYSIG: STD_LOGIC;
signal SENDCLOCKSIG: STD_LOGIC;
signal OLDSENDCLOCKSIG: STD_LOGIC;
signal RECEIVECLOCKSIG: STD_LOGIC;
signal OLDRECEIVECLOCKSIG: STD_LOGIC;
signal RECEIVEDATASIG: STD_LOGIC;
signal RECEIVEDONESIG: STD_LOGIC;
signal OLDRECEIVEDONESIG: STD_LOGIC;
signal DATAOUT: STD_LOGIC_VECTOR(7 downto 0);
begin
    host_interface_clk2: process begin

    wait until CLK'event and CLK=CYCLE2;

-- send ready signal OUT
    READY <= READYSIG;
-- stabalize read signal
    READSIG <= REQUEST;

-- stabalize send clock signal
    SENDCLOCKSIG <= SENDCLOCK;
-- send dataout bit
    SENDDATA <= DATAOUT(0);

-- stabalize receive clock signal
    RECEIVECLOCKSIG <= RECEIVECLOCK;
-- stabalize receive data signal
    RECEIVEDATASIG <= RECEIVEDATA;
-- stabalize receive done signal
    RECEIVEDONESIG <= RECEIVEDONE;

    end process host_interface_clk2;

    host_interface_clk1: process begin

    wait until CLK'event and CLK=CYCLE1;

    if (READSIG=READEND) then
-- if READ at anytime is low, reset read mode
        POP <= '0';
        READYSIG <= '0';
    else
-- read is set, whats the status of the pop?
        if (READYSIG='0') then
-- no pop occured in the last cycle
            if (POPDONE='0') then
-- the data wasn't available this cycle, try again if its ok
                if (OKTOPOP='0') then
```

```
                POP <= '1';
                READYSIG <= '0';
            else
                POP <= '0';
                READYSIG <= '0';
            end if;
        else
-- data is available, get the data, setup the next state
            POP <= '0';
--          DATAOUT <= DATAIN;  -- see below
            READYSIG <= '1';
        end if;
    else
-- if the pop was completed, do nothing to the signals
        POP <= '0';
        READYSIG <= '1';
    end if;
end if;

if (POPDONE='1') then
    DATAOUT <= DATAIN;
elsif ( (not SENDCLOCKSIG)=OLDSENDCLOCKSIG and READYSIG='1') then
    OLDSENDCLOCKSIG <= SENDCLOCKSIG;
    if (SENDCLOCKSIG='1') then
        DATAOUT(6 downto 0) <= DATAOUT(7 downto 1);
    end if;
end if;

end process host_interface_clk1;

host_interface_receive: process begin

wait until CLK'event and CLK=CYCLE1;

if ( (not RECEIVECLOCKSIG)=OLDRECEIVECLOCKSIG) then
    OLDRECEIVECLOCKSIG <= RECEIVECLOCKSIG;
    if (RECEIVECLOCKSIG='1') then
        COMMANDDATAOUT(10) <= RECEIVEDATASIG;
        COMMANDDATAOUT(9 downto 0) <= COMMANDDATAOUT(10 downto 1);
    end if;
end if;

if ( (not RECEIVEDONESIG)=OLDRECEIVEDONESIG) then
    OLDRECEIVEDONESIG <= RECEIVEDONESIG;
    if (RECEIVEDONESIG='1') then
        COMMANDDATAVALID <= '1';
    else
        COMMANDDATAVALID <= '0';
    end if;
else
    COMMANDDATAVALID <= '0';
end if;

end process host_interface_receive;
end behaviour;
```

B.2.4 Video Processor VHDL Code

B.2.4.1 minmax.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity camera_interface is
    port(CLK: in STD_LOGIC;
    CAMERADATA: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(15 downto 0);
    OUTDATA16: out STD_LOGIC;
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;

    COMMANDDATA: in STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: in STD_LOGIC;

    CAPTUREON: in STD_LOGIC;

    PUSH: out STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end camera_interface;

architecture behaviour of camera_interface is
constant CYCLE1: STD_LOGIC := '1';
constant CYCLE2: STD_LOGIC := '0';
signal LVALSIG: STD_LOGIC;
signal OLDLVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal OLDNEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal DELAYCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal STOP: STD_LOGIC;
signal PIXELPROC: STD_LOGIC;
signal COMPPROC: STD_LOGIC;
signal FIRSTPIXEL: STD_LOGIC;
signal LAST: SIGNED(8 downto 0);
signal PIXEL1: UNSIGNED(7 downto 0);
signal PIXEL2: UNSIGNED(7 downto 0);

signal COMPCOUNT: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPVALUE: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPCHANGE: STD_LOGIC;
signal COMPINIT: STD_LOGIC;
signal COMPREPEAT: STD_LOGIC;

signal SYNCCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal SYNCNUMBER: STD_LOGIC_VECTOR(5 downto 0) := "000000";
signal SYNCED: STD_LOGIC;
signal MAXTHRES: UNSIGNED(7 downto 0);
signal MINTHRES: UNSIGNED(7 downto 0);
signal BACKGROUND: UNSIGNED(7 downto 0);
signal LASTBAD: STD_LOGIC;
begin
```

```
-- stabalize incoming signals
    camera_stabalize: process begin
    wait until CLK'event and CLK=CYCLE1;

    PIXEL1 <= CAMERADATA;
    LVALSIG <= LVAL;
    PVALSIG <= PVAL;
    OLDLVALSIG <= LVALSIG;

-- delay line valid for 7 pixel cycles
    if ( not (OLDLVALSIG = LVALSIG) ) then
        DELAYCOUNT <= "000";
    else
        if (DELAYCOUNT < "100") then
            DELAYCOUNT <= DELAYCOUNT + 1;
            NEWLVALSIG <= LVALSIG;
        end if;
    end if;

    end process;

-- when FIFO is full we have to stop immediately
    camera_stop: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (OKTOPUSH='1' and CAPTUREON='1') then
        STOP <= '1';
    elsif (CAPTUREON='0') then
        STOP <= '0';
    end if;

    end process;

-- generate a signal for the next process to indicate if we should process the
--  next pixel
    camera_setpixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    OLDNEWLVALSIG <= NEWLVALSIG;
    if (PVALSIG='1' and NEWLVALSIG='1' and STOP='0' and CAPTUREON='1') then
        PIXELPROC <= '1';
        PIXEL2 <= PIXEL1;
        if (OLDNEWLVALSIG='0') then
            FIRSTPIXEL <= '1';
        else
            FIRSTPIXEL <= '0';
        end if;
    else
        PIXELPROC <= '0';
        FIRSTPIXEL <= '0';
    end if;

    end process;

-- process the pixel
    camera_pixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;
```

```
    if (PIXELPROC='1') then

        COMPPROC <= '1';
        if (FIRSTPIXEL='0') then
            COMPINIT <= '0';
-- is the pixel in the proper range
            if ( (PIXEL2 >= MAXTHRES) or (PIXEL2 <= MINTHRES) ) then
-- this pixel is a defect
                COMPCHANGE <= '1';
                LASTBAD <= '1';
                LAST <= CONV_SIGNED(PIXEL2,9);
            else
-- no defect, send the background
                COMPCHANGE <= LASTBAD;
                LASTBAD <= '0';
                LAST <= CONV_SIGNED(BACKGROUND,9);
            end if;
        else
-- first pixel, initialize compressor
            COMPINIT <= '1';
            COMPCHANGE <= '0';
            LASTBAD <= LASTBAD;
            LAST <= CONV_SIGNED(BACKGROUND,9);
        end if;

    else
-- no pixel to process, make sure the RLE compressor does nothing too
        COMPINIT <= '0';
        COMPPROC <= '0';
        COMPCHANGE <= '0';
        LASTBAD <= LASTBAD;
        LAST <= CONV_SIGNED(BACKGROUND,9);

    end if;

    end process;

-- RLE compressor
    camera_compressor: process begin
    wait until CLK'event and CLK=CYCLE1;

-- wait for 6 inactive cycles before flushing the RLE codes
    if (SYNCCOUNT="110" and SYNCED='0') then
        SYNCCOUNT <= SYNCCOUNT + 1;
        OUTDATA(15 downto 8) <= COMPVALUE;

        PUSH <= '1';
        COMPCOUNT <= "00000001";

        COMPREPEAT <= '0';
        OUTDATA(7 downto 0) <= COMPCOUNT;
        if (COMPREPEAT='1') then
            OUTDATA16 <= '0';
        else
            OUTDATA16 <= '1';
        end if;

        SYNCED <= '0';
```

```
    elsif (COMPPROC='1') then
-- pixel to process, reset synchronization counter
        SYNCCOUNT <= (others => '0');
        OUTDATA(15 downto 8) <= COMPVALUE;

-- Check if repeat count is over 254, use special command code
        if ( (COMPCHANGE='0' ) and (COMPCOUNT="11111110") ) then
            PUSH <= '1';
            COMPCOUNT <= "00000001";

            COMPREPEAT <= '1';
            OUTDATA(7 downto 0) <= "11111111";
            if (COMPREPEAT='1') then
                OUTDATA16 <= '0';
            else
                OUTDATA16 <= '1';
            end if;

        elsif ( COMPCHANGE='1') then
-- Single pixel, flush it immediately
            PUSH <= '1';
            COMPCOUNT <= "00000001";
            COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

            COMPREPEAT <= '0';
            OUTDATA(7 downto 0) <= COMPCOUNT;
            if (COMPREPEAT='1') then
                OUTDATA16 <= '0';
            else
                OUTDATA16 <= '1';
            end if;

        elsif ( COMPINIT='1') then
-- Initialize compressor
            PUSH <= '0';
            COMPCOUNT <= "00000001";
            COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

            COMPREPEAT <= '0';
            OUTDATA(7 downto 0) <= COMPCOUNT;
            OUTDATA16 <= '0'; -- does not matter

        else
-- Update repeat counter
            COMPCOUNT <= COMPCOUNT + 1;
            PUSH <= '0';
            COMPREPEAT <= COMPREPEAT;
            OUTDATA(7 downto 0) <= COMPCOUNT;
            OUTDATA16 <= '0'; -- does not matter
        end if;

        SYNCED <= '0';

    else
-- setup output for synchronization code
        OUTDATA <= "0000000000000000";
        COMPREPEAT <= COMPREPEAT;

        if (CAPTUREON='0') then
```

```
-- if capture not on, reset varaibles
          SYNCNUMBER <= (others => '0');
          PUSH <= '0';
          OUTDATA16 <= '0';
          SYNCED <= '1';
          SYNCCOUNT <= (others => '0');

       else
          SYNCCOUNT <= SYNCCOUNT + 1;
-- on 7th cycle of inactivity and 2nd line or 64
--  output synchronization signal
          if (SYNCCOUNT="111" and SYNCED='0') then
             SYNCNUMBER <= SYNCNUMBER + 1;
             if (SYNCNUMBER="000010") then
                PUSH <= '1';
             else
                PUSH <= '0';
             end if;
             OUTDATA16 <= '1';
             SYNCED <= '1';

          else
             PUSH <= '0';
             OUTDATA16 <= '0';
             SYNCED <= SYNCED;

          end if;

       end if;

    end if;

    end process;

-- move programmable parameters into proper variables
--  do it bit by bit since they are different types with no
--  proper conversion available
    camera_parameters: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (COMMANDDATAVALID='1') then
       if (COMMANDDATA(10 downto 8)="000") then
          BACKGROUND(7) <= COMMANDDATA(7);
          BACKGROUND(6) <= COMMANDDATA(6);
          BACKGROUND(5) <= COMMANDDATA(5);
          BACKGROUND(4) <= COMMANDDATA(4);
          BACKGROUND(3) <= COMMANDDATA(3);
          BACKGROUND(2) <= COMMANDDATA(2);
          BACKGROUND(1) <= COMMANDDATA(1);
          BACKGROUND(0) <= COMMANDDATA(0);
       elsif (COMMANDDATA(10 downto 8)="001") then
          MAXTHRES(7) <= COMMANDDATA(7);
          MAXTHRES(6) <= COMMANDDATA(6);
          MAXTHRES(5) <= COMMANDDATA(5);
          MAXTHRES(4) <= COMMANDDATA(4);
          MAXTHRES(3) <= COMMANDDATA(3);
          MAXTHRES(2) <= COMMANDDATA(2);
          MAXTHRES(1) <= COMMANDDATA(1);
          MAXTHRES(0) <= COMMANDDATA(0);
```

```
        elsif (COMMANDDATA(10 downto 8)="010") then
            MINTHRES(7) <= COMMANDDATA(7);
            MINTHRES(6) <= COMMANDDATA(6);
            MINTHRES(5) <= COMMANDDATA(5);
            MINTHRES(4) <= COMMANDDATA(4);
            MINTHRES(3) <= COMMANDDATA(3);
            MINTHRES(2) <= COMMANDDATA(2);
            MINTHRES(1) <= COMMANDDATA(1);
            MINTHRES(0) <= COMMANDDATA(0);
        end if;
    end if;

    end process;

end behaviour;
```

B.2.4.2 range.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- Define sub-system ports
entity camera_interface is
    port(CLK: in STD_LOGIC;
    CAMERADATA: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(15 downto 0);
    OUTDATA16: out STD_LOGIC;
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;

    COMMANDDATA: in STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: in STD_LOGIC;

    CAPTUREON: in STD_LOGIC;

    PUSH: out STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end camera_interface;

architecture behaviour of camera_interface is
constant CYCLE1: STD_LOGIC := '1';
constant CYCLE2: STD_LOGIC := '0';
signal LVALSIG: STD_LOGIC;
signal OLDLVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal OLDNEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal DELAYCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal STOP: STD_LOGIC;
signal PIXELPROC: STD_LOGIC;
signal COMPPROC: STD_LOGIC;
signal FIRSTPIXEL: STD_LOGIC;
signal LAST: SIGNED(8 downto 0);
signal PIXEL1: UNSIGNED(7 downto 0);
signal PIXEL2: UNSIGNED(7 downto 0);
```

```
signal COMPCOUNT: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPVALUE: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPCHANGE: STD_LOGIC;
signal COMPINIT: STD_LOGIC;
signal COMPREPEAT: STD_LOGIC;

signal SYNCCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal SYNCNUMBER: STD_LOGIC_VECTOR(5 downto 0) := "000000";
signal SYNCED: STD_LOGIC;
signal MAXVALUE: UNSIGNED(7 downto 0);
signal MINVALUE: UNSIGNED(7 downto 0);
signal BACKGROUND: UNSIGNED(7 downto 0);
signal LASTBAD: STD_LOGIC;
begin

-- stabalize incoming signals
    camera_stabalize: process begin
    wait until CLK'event and CLK=CYCLE1;

    PIXEL1 <= CAMERADATA;
    LVALSIG <= LVAL;
    PVALSIG <= PVAL;
    OLDLVALSIG <= LVALSIG;

-- delay line valid for 7 pixel cycles
    if ( not (OLDLVALSIG = LVALSIG) ) then
        DELAYCOUNT <= "000";
    else
        if (DELAYCOUNT < "100") then
            DELAYCOUNT <= DELAYCOUNT + 1;
            NEWLVALSIG <= LVALSIG;
        end if;
    end if;

    end process;

-- when FIFO is full we have to stop immediately
    camera_stop: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (OKTOPUSH='1' and CAPTUREON='1') then
        STOP <= '1';
    elsif (CAPTUREON='0') then
        STOP <= '0';
    end if;

    end process;

-- generate a signal for the next process to indicate if we should process the
--  next pixel
    camera_setpixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    OLDNEWLVALSIG <= NEWLVALSIG;
    if (PVALSIG='1' and NEWLVALSIG='1' and STOP='0' and CAPTUREON='1') then
        PIXELPROC <= '1';
        PIXEL2 <= PIXEL1;
        if (OLDNEWLVALSIG='0') then
            FIRSTPIXEL <= '1';
```

```
        else
            FIRSTPIXEL <= '0';
        end if;
    else
        PIXELPROC <= '0';
        FIRSTPIXEL <= '0';
    end if;

    end process;

-- process the pixel
    camera_pixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (PIXELPROC='1') then

        COMPPROC <= '1';
        if (FIRSTPIXEL='0') then
            COMPINIT <= '0';
-- is the pixel in the proper range
-- similar to maxmin, but signs are different
            if ( (PIXEL2 <= MAXVALUE) and (PIXEL2 >= MINVALUE) ) then
-- this pixel is a defect
                COMPCHANGE <= '1';
                LASTBAD <= '1';
                LAST <= CONV_SIGNED(PIXEL2,9);
            else
-- no defect, send the background
                COMPCHANGE <= LASTBAD;
                LASTBAD <= '0';
                LAST <= CONV_SIGNED(BACKGROUND,9);
            end if;
        else
-- first pixel, initialize compressor
            COMPINIT <= '1';
            COMPCHANGE <= '0';
            LASTBAD <= LASTBAD;
            LAST <= CONV_SIGNED(BACKGROUND,9);
        end if;

    else
-- no pixel to process, make sure the RLE compressor does nothing too
        COMPINIT <= '0';
        COMPPROC <= '0';
        COMPCHANGE <= '0';
        LASTBAD <= LASTBAD;
        LAST <= CONV_SIGNED(BACKGROUND,9);

    end if;

    end process;

-- RLE compressor
    camera_compressor: process begin
    wait until CLK'event and CLK=CYCLE1;

-- wait for 6 inactive cycles before flushing the RLE codes
    if (SYNCCOUNT="110" and SYNCED='0') then
        SYNCCOUNT <= SYNCCOUNT + 1;
```

```
        OUTDATA(15 downto 8) <= COMPVALUE;

        PUSH <= '1';
        COMPCOUNT <= "00000001";

        COMPREPEAT <= '0';
        OUTDATA(7 downto 0) <= COMPCOUNT;
        if (COMPREPEAT='1') then
           OUTDATA16 <= '0';
        else
           OUTDATA16 <= '1';
        end if;

        SYNCED <= '0';

    elsif (COMPPROC='1') then
-- pixel to process, reset synchronization counter
        SYNCCOUNT <= (others => '0');
        OUTDATA(15 downto 8) <= COMPVALUE;

-- Check if repeat count is over 254, use special command code
        if ( (COMPCHANGE='0' ) and (COMPCOUNT="11111110") ) then
           PUSH <= '1';
           COMPCOUNT <= "00000001";

           COMPREPEAT <= '1';
           OUTDATA(7 downto 0) <= "11111111";
           if (COMPREPEAT='1') then
              OUTDATA16 <= '0';
           else
              OUTDATA16 <= '1';
           end if;

        elsif ( COMPCHANGE='1') then
-- Single pixel, flush it immediately
           PUSH <= '1';
           COMPCOUNT <= "00000001";
           COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

           COMPREPEAT <= '0';
           OUTDATA(7 downto 0) <= COMPCOUNT;
           if (COMPREPEAT='1') then
              OUTDATA16 <= '0';
           else
              OUTDATA16 <= '1';
           end if;

        elsif ( COMPINIT='1') then
-- Initialize compressor
           PUSH <= '0';
           COMPCOUNT <= "00000001";
           COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

           COMPREPEAT <= '0';
           OUTDATA(7 downto 0) <= COMPCOUNT;
           OUTDATA16 <= '0'; -- does not matter

        else
-- Update repeat counter
```

```
            COMPCOUNT <= COMPCOUNT + 1;
            PUSH <= '0';
            COMPREPEAT <= COMPREPEAT;
            OUTDATA(7 downto 0) <= COMPCOUNT;
            OUTDATA16 <= '0'; -- does not matter
        end if;

        SYNCED <= '0';

    else
-- setup output for synchronization code
        OUTDATA <= "0000000000000000";
        COMPREPEAT <= COMPREPEAT;

        if (CAPTUREON='0') then
-- if capture not on, reset varaibles
            SYNCNUMBER <= (others => '0');
            PUSH <= '0';
            OUTDATA16 <= '0';
            SYNCED <= '1';
            SYNCCOUNT <= (others => '0');

        else
            SYNCCOUNT <= SYNCCOUNT + 1;
-- on 7th cycle of inactivity and 2nd line or 64
--  output synchronization signal
            if (SYNCCOUNT="111" and SYNCED='0') then
                SYNCNUMBER <= SYNCNUMBER + 1;
                if (SYNCNUMBER="000010") then
                    PUSH <= '1';
                else
                    PUSH <= '0';
                end if;
                OUTDATA16 <= '1';
                SYNCED <= '1';

            else
                PUSH <= '0';
                OUTDATA16 <= '0';
                SYNCED <= SYNCED;

            end if;

        end if;

    end if;

    end process;

-- move programmable parameters into proper variables
--  do it bit by bit since they are different types with no
--  proper conversion available
    camera_parameters: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (COMMANDDATAVALID='1') then
        if (COMMANDDATA(10 downto 8)="000") then
            BACKGROUND(7) <= COMMANDDATA(7);
            BACKGROUND(6) <= COMMANDDATA(6);
```

```
            BACKGROUND(5) <= COMMANDDATA(5);
            BACKGROUND(4) <= COMMANDDATA(4);
            BACKGROUND(3) <= COMMANDDATA(3);
            BACKGROUND(2) <= COMMANDDATA(2);
            BACKGROUND(1) <= COMMANDDATA(1);
            BACKGROUND(0) <= COMMANDDATA(0);
        elsif (COMMANDDATA(10 downto 8)="001") then
            MAXVALUE(7) <= COMMANDDATA(7);
            MAXVALUE(6) <= COMMANDDATA(6);
            MAXVALUE(5) <= COMMANDDATA(5);
            MAXVALUE(4) <= COMMANDDATA(4);
            MAXVALUE(3) <= COMMANDDATA(3);
            MAXVALUE(2) <= COMMANDDATA(2);
            MAXVALUE(1) <= COMMANDDATA(1);
            MAXVALUE(0) <= COMMANDDATA(0);
        elsif (COMMANDDATA(10 downto 8)="010") then
            MINVALUE(7) <= COMMANDDATA(7);
            MINVALUE(6) <= COMMANDDATA(6);
            MINVALUE(5) <= COMMANDDATA(5);
            MINVALUE(4) <= COMMANDDATA(4);
            MINVALUE(3) <= COMMANDDATA(3);
            MINVALUE(2) <= COMMANDDATA(2);
            MINVALUE(1) <= COMMANDDATA(1);
            MINVALUE(0) <= COMMANDDATA(0);
        end if;
    end if;

    end process;

end behaviour;
```

B.2.4.3 threshold.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- Define sub-system ports
entity camera_interface is
    port(CLK: in STD_LOGIC;
    CAMERADATA: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(15 downto 0);
    OUTDATA16: out STD_LOGIC;
    LVAL: in STD_LOGIC;
    FVAL: in STD_LOGIC;

    COMMANDDATA: in STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: in STD_LOGIC;

    CAPTUREON: in STD_LOGIC;

    PUSH: out STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end camera_interface;

architecture behaviour of camera_interface is
constant CYCLE1: STD_LOGIC := '1';
```

```
constant CYCLE2: STD_LOGIC := '0';
signal LVALSIG: STD_LOGIC;
signal OLDLVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal OLDNEWLVALSIG: STD_LOGIC;
signal FVALSIG: STD_LOGIC;
signal DELAYCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal STOP: STD_LOGIC;
signal PIXELPROC: STD_LOGIC;
signal COMPPROC: STD_LOGIC;
signal FIRSTPIXEL: STD_LOGIC;
signal LAST: SIGNED(8 downto 0);
signal PIXEL1: UNSIGNED(7 downto 0);
signal PIXEL2: UNSIGNED(7 downto 0);

signal COMPCOUNT: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPVALUE: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPCHANGE: STD_LOGIC;
signal COMPINIT: STD_LOGIC;
signal COMPREPEAT: STD_LOGIC;

signal SYNCCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal SYNCNUMBER: STD_LOGIC_VECTOR(5 downto 0) := "000000";
signal SYNCED: STD_LOGIC;
signal THRES: UNSIGNED(7 downto 0);
begin

-- stabalize incoming signals
    camera_stabalize: process begin
    wait until CLK'event and CLK=CYCLE1;

    PIXEL1 <= CAMERADATA;
    LVALSIG <= LVAL;
    FVALSIG <= FVAL;
    OLDLVALSIG <= LVALSIG;

-- delay line valid for 7 pixel cycles
    if ( not (OLDLVALSIG = LVALSIG) ) then
        DELAYCOUNT <= "000";
    else
        if (DELAYCOUNT < "100") then
            DELAYCOUNT <= DELAYCOUNT + 1;
            NEWLVALSIG <= LVALSIG;
        end if;
    end if;

    end process;

-- when FIFO is full we have to stop immediately
    camera_stop: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (OKTOPUSH='1' and CAPTUREON='1') then
        STOP <= '1';
    elsif (CAPTUREON='0') then
        STOP <= '0';
    end if;

    end process;
```

```
-- generate a signal for the next process to indicate if we should process the
--  next pixel
    camera_setpixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    OLDNEWLVALSIG <= NEWLVALSIG;
    if (PVALSIG='1' and NEWLVALSIG='1' and STOP='0' and CAPTUREON='1') then
        PIXELPROC <= '1';
        PIXEL2 <= PIXEL1;
        if (OLDNEWLVALSIG='0') then
            FIRSTPIXEL <= '1';
        else
            FIRSTPIXEL <= '0';
        end if;
    else
        PIXELPROC <= '0';
        FIRSTPIXEL <= '0';
    end if;

    end process;

-- process the pixel
    camera_pixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (PIXELPROC='1') then

        COMPPROC <= '1';
        LAST <= CONV_SIGNED(PIXEL2,9);
        if (FIRSTPIXEL='0') then
            COMPINIT <= '0';
            if ( ABS ( LAST - PIXEL2 ) >= THRES ) then
-- this pixel is a defect
                COMPCHANGE <= '1';
            else
-- no defect, send the background
                COMPCHANGE <= '0';
            end if;
        else
-- first pixel, initialize compressor
            COMPINIT <= '1';
            COMPCHANGE <= '0';
        end if;

    else
-- no pixel to process, make sure the RLE compressor does nothing too
        COMPINIT <= '0';
        COMPPROC <= '0';
        COMPCHANGE <= '0';

    end if;

    end process;

-- RLE compressor
    camera_compressor: process begin
    wait until CLK'event and CLK=CYCLE1;
```

```
-- wait for 6 inactive cycles before flushing the RLE codes
   if (SYNCCOUNT="110" and SYNCED='0') then
      SYNCCOUNT <= SYNCCOUNT + 1;
      OUTDATA(15 downto 8) <= COMPVALUE;

      PUSH <= '1';
      COMPCOUNT <= "00000001";

      COMPREPEAT <= '0';
      OUTDATA(7 downto 0) <= COMPCOUNT;
      if (COMPREPEAT='1') then
         OUTDATA16 <= '0';
      else
         OUTDATA16 <= '1';
      end if;

      SYNCED <= '0';

   elsif (COMPPROC='1') then
-- pixel to process, reset synchronization counter
      SYNCCOUNT <= (others => '0');
      OUTDATA(15 downto 8) <= COMPVALUE;

-- Check if repeat count is over 254, use special command code
      if ( (COMPCHANGE='0' ) and (COMPCOUNT="11111110") ) then
         PUSH <= '1';
         COMPCOUNT <= "00000001";

         COMPREPEAT <= '1';
         OUTDATA(7 downto 0) <= "11111111";
         if (COMPREPEAT='1') then
            OUTDATA16 <= '0';
         else
            OUTDATA16 <= '1';
         end if;

      elsif ( COMPCHANGE='1') then
-- Single pixel, flush it immediately
         PUSH <= '1';
         COMPCOUNT <= "00000001";
         COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

         COMPREPEAT <= '0';
         OUTDATA(7 downto 0) <= COMPCOUNT;
         if (COMPREPEAT='1') then
            OUTDATA16 <= '0';
         else
            OUTDATA16 <= '1';
         end if;

      elsif ( COMPINIT='1') then
-- Initialize compressor
         PUSH <= '0';
         COMPCOUNT <= "00000001";
         COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

         COMPREPEAT <= '0';
         OUTDATA(7 downto 0) <= COMPCOUNT;
         OUTDATA16 <= '0'; -- does not matter
```

```
        else
-- Update repeat counter
            COMPCOUNT <= COMPCOUNT + 1;
            PUSH <= '0';
            COMPREPEAT <= COMPREPEAT;
            OUTDATA(7 downto 0) <= COMPCOUNT;
            OUTDATA16 <= '0'; -- does not matter
        end if;

        SYNCED <= '0';

    else
-- setup output for synchronization code
        OUTDATA <= "0000000000000000";
        COMPREPEAT <= COMPREPEAT;

        if (CAPTUREON='0') then
-- if capture not on, reset varaibles
            SYNCNUMBER <= (others => '0');
            PUSH <= '0';
            OUTDATA16 <= '0';
            SYNCED <= '1';
            SYNCCOUNT <= (others => '0');

        else
            SYNCCOUNT <= SYNCCOUNT + 1;
-- on 7th cycle of inactivity and 2nd line or 64
--  output synchronization signal
            if (SYNCCOUNT="111" and SYNCED='0') then
                SYNCNUMBER <= SYNCNUMBER + 1;
                if (SYNCNUMBER="000010") then
                    PUSH <= '1';
                else
                    PUSH <= '0';
                end if;
                OUTDATA16 <= '1';
                SYNCED <= '1';

            else
                PUSH <= '0';
                OUTDATA16 <= '0';
                SYNCED <= SYNCED;

            end if;

        end if;

    end if;

    end process;

-- move programmable parameters into proper variables
--  do it bit by bit since they are different types with no
--  proper conversion available
    camera_parameters: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (COMMANDDATAVALID='1') then
```

```
        if (COMMANDDATA(10 downto 8)="000") then
            THRES(7) <= COMMANDDATA(7);
            THRES(6) <= COMMANDDATA(6);
            THRES(5) <= COMMANDDATA(5);
            THRES(4) <= COMMANDDATA(4);
            THRES(3) <= COMMANDDATA(3);
            THRES(2) <= COMMANDDATA(2);
            THRES(1) <= COMMANDDATA(1);
            THRES(0) <= COMMANDDATA(0);
        end if;
    end if;

    end process;

end behaviour;
```

B.2.4.4 deltatracker.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity camera_interface is
    port(CLK: in STD_LOGIC;
    CAMERADATA: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(15 downto 0);
    OUTDATA16: out STD_LOGIC;
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;

    COMMANDDATA: in STD_LOGIC_VECTOR(10 downto 0);
    COMMANDDATAVALID: in STD_LOGIC;

    CAPTUREON: in STD_LOGIC;

    PUSH: out STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end camera_interface;

architecture behaviour of camera_interface is
constant CYCLE1: STD_LOGIC := '1';
constant CYCLE2: STD_LOGIC := '0';
signal LVALSIG: STD_LOGIC;
signal OLDLVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal OLDNEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal DELAYCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal STOP: STD_LOGIC;
signal PIXELPROC: STD_LOGIC;
signal COMPPROC: STD_LOGIC;
signal FIRSTPIXEL: STD_LOGIC;
signal LAST: SIGNED(8 downto 0);
signal PIXEL1: UNSIGNED(7 downto 0);
signal PIXEL2: UNSIGNED(7 downto 0);
```

```
signal COMPCOUNT: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPVALUE: STD_LOGIC_VECTOR(7 downto 0) := "00000000";
signal COMPCHANGE: STD_LOGIC;
signal COMPINIT: STD_LOGIC;
signal COMPREPEAT: STD_LOGIC;

signal SYNCCOUNT: STD_LOGIC_VECTOR(2 downto 0) := "000";
signal SYNCNUMBER: STD_LOGIC_VECTOR(5 downto 0) := "000000";
signal SYNCED: STD_LOGIC;
signal THRES: UNSIGNED(7 downto 0);
signal BACKGROUND: UNSIGNED(7 downto 0);
signal LASTBAD: STD_LOGIC;
signal BACKVAL: UNSIGNED(7 downto 0);
signal BACKCOMP: UNSIGNED(7 downto 0);
signal LOWC: UNSIGNED(3 downto 0);
signal LOWCBW: UNSIGNED(3 downto 0);
signal BACKPROC: STD_LOGIC;
begin

-- stabalize incoming signals
    camera_stabalize: process begin
    wait until CLK'event and CLK=CYCLE1;

    PIXEL1 <= CAMERADATA;
    LVALSIG <= LVAL;
    PVALSIG <= PVAL;
    OLDLVALSIG <= LVALSIG;

-- delay line valid for 7 pixel cycles
    if ( not (OLDLVALSIG = LVALSIG) ) then
        DELAYCOUNT <= "000";
    else
        if (DELAYCOUNT < "100") then
            DELAYCOUNT <= DELAYCOUNT + 1;
            NEWLVALSIG <= LVALSIG;
        end if;
    end if;

    end process;

-- when FIFO is full we have to stop immediately
    camera_stop: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (OKTOPUSH='1' and CAPTUREON='1') then
        STOP <= '1';
    elsif (CAPTUREON='0') then
        STOP <= '0';
    end if;

    end process;

-- generate a signal for the next process to indicate if we should process the
--  next pixel
    camera_setpixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    OLDNEWLVALSIG <= NEWLVALSIG;
    if (PVALSIG='1' and NEWLVALSIG='1' and STOP='0' and CAPTUREON='1') then
```

```
        PIXELPROC <= '1';
        PIXEL2 <= PIXEL1;
        if (OLDNEWLVALSIG='0') then
           FIRSTPIXEL <= '1';
        else
           FIRSTPIXEL <= '0';
        end if;
    else
        PIXELPROC <= '0';
        FIRSTPIXEL <= '0';
    end if;

    end process;

-- process the pixel
    camera_pixelproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (PIXELPROC='1') then

        BACKVAL <= PIXEL2;
        COMPPROC <= '1';
        if (FIRSTPIXEL='0') then
           COMPINIT <= '0';
-- does the slope exceed the programmable threshold
           if ( (PIXEL2 >= (BACKCOMP+THRES)) or (PIXEL2 <= (BACKCOMP-THRES)) )
then
-- this pixel is a defect
              COMPCHANGE <= '1';
              LASTBAD <= '1';
              BACKPROC <= '0';
              LAST <= CONV_SIGNED(PIXEL2,9);
           else
-- no defect, send the background
              COMPCHANGE <= LASTBAD;
              LASTBAD <= '0';
              BACKPROC <= '1';
              LAST <= CONV_SIGNED(BACKGROUND,9);
           end if;
        else
-- first pixel, initialize compressor and deltatracker
           COMPINIT <= '1';
           COMPCHANGE <= '0';
           LASTBAD <= LASTBAD;
           BACKPROC <= '0';
           LAST <= CONV_SIGNED(BACKGROUND,9);
        end if;

    else
-- no pixel to process, make sure the RLE compressor does nothing too
        COMPINIT <= '0';
        COMPPROC <= '0';
        COMPCHANGE <= '0';
        LASTBAD <= LASTBAD;
        BACKPROC <= '0';
        LAST <= CONV_SIGNED(BACKGROUND,9);

    end if;
```

```
    end process;

-- process deltatracker variables
    camera_backproc: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (BACKPROC = '1') then
-- process background
        if (BACKVAL > BACKCOMP) then
            if (LOWC = LOWCBW) then
-- increment background
                LOWC <= (others => '0');
                if (not (BACKCOMP = 255)) then
                    BACKCOMP <= BACKCOMP + 1;
                end if;
            else
                LOWC <= LOWC + 1;
            end if;
        elsif (BACKVAL < BACKCOMP) then
            if (LOWC = 0) then
-- decrement background
                LOWC <= LOWCBW;
                if (not (BACKCOMP = 0)) then
                    BACKCOMP <= BACKCOMP - 1;
                end if;
            else
                LOWC <= LOWC - 1;
            end if;
        end if;
    elsif (FIRSTPIXEL='1') then
        LOWC <= (others => '0');
        BACKCOMP <= PIXEL2;
    end if;

    end process;

-- RLE compressor
    camera_compressor: process begin
    wait until CLK'event and CLK=CYCLE1;

-- wait for 6 inactive cycles before flushing the RLE codes
    if (SYNCCOUNT="110" and SYNCED='0') then
        SYNCCOUNT <= SYNCCOUNT + 1;
        OUTDATA(15 downto 8) <= COMPVALUE;

        PUSH <= '1';
        COMPCOUNT <= "00000001";

        COMPREPEAT <= '0';
        OUTDATA(7 downto 0) <= COMPCOUNT;
        if (COMPREPEAT='1') then
            OUTDATA16 <= '0';
        else
            OUTDATA16 <= '1';
        end if;

        SYNCED <= '0';

    elsif (COMPPROC='1') then
```

```
-- pixel to process, reset synchronization counter
       SYNCCOUNT <= (others => '0');
       OUTDATA(15 downto 8) <= COMPVALUE;

-- Check if repeat count is over 254, use special command code
       if ( (COMPCHANGE='0' ) and (COMPCOUNT="11111110") ) then
           PUSH <= '1';
           COMPCOUNT <= "00000001";

           COMPREPEAT <= '1';
           OUTDATA(7 downto 0) <= "11111111";
           if (COMPREPEAT='1') then
              OUTDATA16 <= '0';
           else
              OUTDATA16 <= '1';
           end if;

       elsif ( COMPCHANGE='1') then
-- Single pixel, flush it immediately
           PUSH <= '1';
           COMPCOUNT <= "00000001";
           COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

           COMPREPEAT <= '0';
           OUTDATA(7 downto 0) <= COMPCOUNT;
           if (COMPREPEAT='1') then
              OUTDATA16 <= '0';
           else
              OUTDATA16 <= '1';
           end if;

       elsif ( COMPINIT='1') then
-- Initialize compressor
           PUSH <= '0';
           COMPCOUNT <= "00000001";
           COMPVALUE <= CONV_STD_LOGIC_VECTOR(LAST,8);

           COMPREPEAT <= '0';
           OUTDATA(7 downto 0) <= COMPCOUNT;
           OUTDATA16 <= '0'; -- does not matter

       else
-- Update repeat counter
           COMPCOUNT <= COMPCOUNT + 1;
           PUSH <= '0';
           COMPREPEAT <= COMPREPEAT;
           OUTDATA(7 downto 0) <= COMPCOUNT;
           OUTDATA16 <= '0'; -- does not matter
       end if;

       SYNCED <= '0';

    else
-- setup output for synchronization code
       OUTDATA <= "0000000000000000";
       COMPREPEAT <= COMPREPEAT;

       if (CAPTUREON='0') then
-- if capture not on, reset varaibles
```

```
            SYNCNUMBER <= (others => '0');
            PUSH <= '0';
            OUTDATA16 <= '0';
            SYNCED <= '1';
            SYNCCOUNT <= (others => '0');

        else
            SYNCCOUNT <= SYNCCOUNT + 1;
-- on 7th cycle of inactivity and 2nd line or 64
--  output synchronization signal
            if (SYNCCOUNT="111" and SYNCED='0') then
                SYNCNUMBER <= SYNCNUMBER + 1;
                if (SYNCNUMBER="000010") then
                    PUSH <= '1';
                else
                    PUSH <= '0';
                end if;
                OUTDATA16 <= '1';
                SYNCED <= '1';

            else
                PUSH <= '0';
                OUTDATA16 <= '0';
                SYNCED <= SYNCED;

            end if;

        end if;

    end if;

    end process;

-- move programmable parameters into proper variables
--  do it bit by bit since they are different types with no
--  proper conversion available

    camera_parameters: process begin
    wait until CLK'event and CLK=CYCLE1;

    if (COMMANDDATAVALID='1') then
        if (COMMANDDATA(10 downto 8)="000") then
            BACKGROUND(7) <= COMMANDDATA(7);
            BACKGROUND(6) <= COMMANDDATA(6);
            BACKGROUND(5) <= COMMANDDATA(5);
            BACKGROUND(4) <= COMMANDDATA(4);
            BACKGROUND(3) <= COMMANDDATA(3);
            BACKGROUND(2) <= COMMANDDATA(2);
            BACKGROUND(1) <= COMMANDDATA(1);
            BACKGROUND(0) <= COMMANDDATA(0);
        elsif (COMMANDDATA(10 downto 8)="001") then
            THRES(7) <= COMMANDDATA(7);
            THRES(6) <= COMMANDDATA(6);
            THRES(5) <= COMMANDDATA(5);
            THRES(4) <= COMMANDDATA(4);
            THRES(3) <= COMMANDDATA(3);
            THRES(2) <= COMMANDDATA(2);
            THRES(1) <= COMMANDDATA(1);
            THRES(0) <= COMMANDDATA(0);
```

```
        elsif (COMMANDDATA(10 downto 8)="010") then
            LOWCBW(3) <= COMMANDDATA(3);
            LOWCBW(2) <= COMMANDDATA(2);
            LOWCBW(1) <= COMMANDDATA(1);
            LOWCBW(0) <= COMMANDDATA(0);
        end if;
    end if;

    end process;

end behaviour;
```

B.3 PC Host Software Code

B.3.1 Makefile

The makefile or project files describe the how all the source code modules connect with each other.

B.3.1.1 larch.mak

```
# Microsoft Developer Studio Generated NMAKE File, Format Version 40001
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Application" 0x0101

!IF "$(CFG)" == ""
CFG=larch - Win32 Debug
!MESSAGE No configuration specified.  Defaulting to larch - Win32 Debug.
!ENDIF

!IF "$(CFG)" != "larch - Win32 Release" && "$(CFG)" != "larch - Win32 Debug"
!MESSAGE Invalid configuration "$(CFG)" specified.
!MESSAGE You can specify a configuration when running NMAKE on this makefile
!MESSAGE by defining the macro CFG on the command line.  For example:
!MESSAGE
!MESSAGE NMAKE /f "larch.mak" CFG="larch - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "larch - Win32 Release" (based on "Win32 (x86) Application")
!MESSAGE "larch - Win32 Debug" (based on "Win32 (x86) Application")
!MESSAGE
!ERROR An invalid configuration is specified.
!ENDIF

!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
################################################################################
# Begin Project
# PROP Target_Last_Scanned "larch - Win32 Debug"
CPP=cl.exe
RSC=rc.exe
MTL=mktyplib.exe

!IF  "$(CFG)" == "larch - Win32 Release"

# PROP BASE Use_MFC 6
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 6
# PROP Use_Debug_Libraries 0
```

```
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
OUTDIR=.\Release
INTDIR=.\Release

ALL : "$(OUTDIR)\larch.exe" "$(OUTDIR)\larch.hlp"

CLEAN :
    -@erase ".\Release\larch.exe"
    -@erase ".\Release\Serial.obj"
    -@erase ".\Release\larch.pch"
    -@erase ".\Release\ChildFrm.obj"
    -@erase ".\Release\larchCptDoc.obj"
    -@erase ".\Release\GetParams.obj"
    -@erase ".\Release\Comp.obj"
    -@erase ".\Release\MainFrm.obj"
    -@erase ".\Release\UnComp.obj"
    -@erase ".\Release\larchCptView.obj"
    -@erase ".\Release\CamInt.obj"
    -@erase ".\Release\StdAfx.obj"
    -@erase ".\Release\comp1.obj"
    -@erase ".\Release\larch.obj"
    -@erase ".\Release\larch.res"
    -@erase ".\Release\larch.hlp"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_AFXDLL" /D "_MBCS" /Yu"stdafx.h" /c
# ADD CPP /nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_AFXDLL" /D "_MBCS" /Yu"stdafx.h" /c
CPP_PROJ=/nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D\
 "_AFXDLL" /D "_MBCS" /Fp"$(INTDIR)/larch.pch" /Yu"stdafx.h" /Fo"$(INTDIR)/" /c
CPP_OBJS=.\Release/
CPP_SBRS=
# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /win32
MTL_PROJ=/nologo /D "NDEBUG" /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG" /d "_AFXDLL"
# ADD RSC /l 0x409 /d "NDEBUG" /d "_AFXDLL"
RSC_PROJ=/l 0x409 /fo"$(INTDIR)/larch.res" /d "NDEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o"$(OUTDIR)/larch.bsc"
BSC32_SBRS=
LINK32=link.exe
# ADD BASE LINK32 /nologo /subsystem:windows /machine:I386
# ADD LINK32 /nologo /subsystem:windows /machine:I386
LINK32_FLAGS=/nologo /subsystem:windows /incremental:no\
 /pdb:"$(OUTDIR)/larch.pdb" /machine:I386 /out:"$(OUTDIR)/larch.exe"
LINK32_OBJS= \
    "$(INTDIR)/Serial.obj" \
    "$(INTDIR)/ChildFrm.obj" \
    "$(INTDIR)/larchCptDoc.obj" \
    "$(INTDIR)/GetParams.obj" \
    "$(INTDIR)/Comp.obj" \
```

```
    "$(INTDIR)/MainFrm.obj" \
    "$(INTDIR)/UnComp.obj" \
    "$(INTDIR)/larchCptView.obj" \
    "$(INTDIR)/CamInt.obj" \
    "$(INTDIR)/StdAfx.obj" \
    "$(INTDIR)/comp1.obj" \
    "$(INTDIR)/larch.obj" \
    "$(INTDIR)/larch.res"

"$(OUTDIR)\larch.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

# PROP BASE Use_MFC 6
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 6
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
OUTDIR=.\Debug
INTDIR=.\Debug

ALL : "$(OUTDIR)\larch.exe" "$(OUTDIR)\larch.bsc" "$(OUTDIR)\larch.hlp"

CLEAN :
    -@erase ".\Debug\vc40.pdb"
    -@erase ".\Debug\larch.pch"
    -@erase ".\Debug\vc40.idb"
    -@erase ".\Debug\larch.bsc"
    -@erase ".\Debug\ChildFrm.sbr"
    -@erase ".\Debug\larchCptView.sbr"
    -@erase ".\Debug\UnComp.sbr"
    -@erase ".\Debug\CamInt.sbr"
    -@erase ".\Debug\larchCptDoc.sbr"
    -@erase ".\Debug\StdAfx.sbr"
    -@erase ".\Debug\Comp.sbr"
    -@erase ".\Debug\MainFrm.sbr"
    -@erase ".\Debug\comp1.sbr"
    -@erase ".\Debug\larch.sbr"
    -@erase ".\Debug\Serial.sbr"
    -@erase ".\Debug\GetParams.sbr"
    -@erase ".\Debug\larch.exe"
    -@erase ".\Debug\Serial.obj"
    -@erase ".\Debug\GetParams.obj"
    -@erase ".\Debug\ChildFrm.obj"
    -@erase ".\Debug\larchCptView.obj"
    -@erase ".\Debug\UnComp.obj"
    -@erase ".\Debug\CamInt.obj"
    -@erase ".\Debug\larchCptDoc.obj"
    -@erase ".\Debug\StdAfx.obj"
    -@erase ".\Debug\Comp.obj"
    -@erase ".\Debug\MainFrm.obj"
```

```
    -@erase ".\Debug\compl.obj"
    -@erase ".\Debug\larch.obj"
    -@erase ".\Debug\larch.res"
    -@erase ".\Debug\larch.ilk"
    -@erase ".\Debug\larch.pdb"
    -@erase ".\Debug\larch.hlp"

"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"

# ADD BASE CPP /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_AFXDLL" /D "_MBCS" /Yu"stdafx.h" /c
# ADD CPP /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
D "_AFXDLL" /D "_MBCS" /FR /Yu"stdafx.h" /c
CPP_PROJ=/nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"\
 /D "_AFXDLL" /D "_MBCS" /FR"$(INTDIR)/" /Fp"$(INTDIR)/larch.pch" /Yu"stdafx.h"\
 /Fo"$(INTDIR)/" /Fd"$(INTDIR)/" /c
CPP_OBJS=.\Debug/
CPP_SBRS=.\Debug/
# ADD BASE MTL /nologo /D "_DEBUG" /win32
# ADD MTL /nologo /D "_DEBUG" /win32
MTL_PROJ=/nologo /D "_DEBUG" /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG" /d "_AFXDLL"
# ADD RSC /l 0x409 /d "_DEBUG" /d "_AFXDLL"
RSC_PROJ=/l 0x409 /fo"$(INTDIR)/larch.res" /d "_DEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o"$(OUTDIR)/larch.bsc"
BSC32_SBRS= \
    "$(INTDIR)/ChildFrm.sbr" \
    "$(INTDIR)/larchCptView.sbr" \
    "$(INTDIR)/UnComp.sbr" \
    "$(INTDIR)/CamInt.sbr" \
    "$(INTDIR)/larchCptDoc.sbr" \
    "$(INTDIR)/StdAfx.sbr" \
    "$(INTDIR)/Comp.sbr" \
    "$(INTDIR)/MainFrm.sbr" \
    "$(INTDIR)/compl.sbr" \
    "$(INTDIR)/larch.sbr" \
    "$(INTDIR)/Serial.sbr" \
    "$(INTDIR)/GetParams.sbr"

"$(OUTDIR)\larch.bsc" : "$(OUTDIR)" $(BSC32_SBRS)
    $(BSC32) @<<
  $(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=link.exe
# ADD BASE LINK32 /nologo /subsystem:windows /debug /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib /nologo /
subsystem:windows /debug /machine:I386
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
 advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib /nologo\
 /subsystem:windows /incremental:yes /pdb:"$(OUTDIR)/larch.pdb" /debug\
 /machine:I386 /out:"$(OUTDIR)/larch.exe"
LINK32_OBJS= \
    "$(INTDIR)/Serial.obj" \
```

```
    "$(INTDIR)/GetParams.obj" \
    "$(INTDIR)/ChildFrm.obj" \
    "$(INTDIR)/larchCptView.obj" \
    "$(INTDIR)/UnComp.obj" \
    "$(INTDIR)/CamInt.obj" \
    "$(INTDIR)/larchCptDoc.obj" \
    "$(INTDIR)/StdAfx.obj" \
    "$(INTDIR)/Comp.obj" \
    "$(INTDIR)/MainFrm.obj" \
    "$(INTDIR)/comp1.obj" \
    "$(INTDIR)/larch.obj" \
    "$(INTDIR)/larch.res"

"$(OUTDIR)\larch.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ENDIF

.c{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.c{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_SBRS)}.sbr:
   $(CPP) $(CPP_PROJ) $<

################################################################################
# Begin Target

# Name "larch - Win32 Release"
# Name "larch - Win32 Debug"

!IF  "$(CFG)" == "larch - Win32 Release"

!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

!ENDIF

################################################################################
# Begin Source File

SOURCE=.\ReadMe.txt

!IF  "$(CFG)" == "larch - Win32 Release"

!ELSEIF  "$(CFG)" == "larch - Win32 Debug"
```

```
!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\StdAfx.cpp
DEP_CPP_STDAF=\
    ".\StdAfx.h"\


!IF  "$(CFG)" == "larch - Win32 Release"

# ADD CPP /Yc"stdafx.h"

BuildCmds= \
    $(CPP) /nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D\
 "_AFXDLL" /D "_MBCS" /Fp"$(INTDIR)/larch.pch" /Yc"stdafx.h" /Fo"$(INTDIR)/" /c\
 $(SOURCE) \


"$(INTDIR)\StdAfx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

"$(INTDIR)\larch.pch" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

# ADD CPP /Yc"stdafx.h"

BuildCmds= \
    $(CPP) /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"\
 /D "_AFXDLL" /D "_MBCS" /FR"$(INTDIR)/" /Fp"$(INTDIR)/larch.pch" /Yc"stdafx.h"\
 /Fo"$(INTDIR)/" /Fd"$(INTDIR)/" /c $(SOURCE) \


"$(INTDIR)\StdAfx.obj" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

"$(INTDIR)\StdAfx.sbr" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

"$(INTDIR)\larch.pch" : $(SOURCE) $(DEP_CPP_STDAF) "$(INTDIR)"
   $(BuildCmds)

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\MainFrm.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_MAINF=\
    ".\StdAfx.h"\
    ".\larch.h"\
```

```
    ".\MainFrm.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\MainFrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_MAINF=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\MainFrm.h"\


"$(INTDIR)\MainFrm.obj" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\MainFrm.sbr" : $(SOURCE) $(DEP_CPP_MAINF) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\ChildFrm.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_CHILD=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\ChildFrm.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\ChildFrm.obj" : $(SOURCE) $(DEP_CPP_CHILD) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_CHILD=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\ChildFrm.h"\


"$(INTDIR)\ChildFrm.obj" : $(SOURCE) $(DEP_CPP_CHILD) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"
```

```
"$(INTDIR)\ChildFrm.sbr" : $(SOURCE) $(DEP_CPP_CHILD) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\larchCptDoc.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_LARCH=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\larchCptDoc.h"\
    ".\larchCptView.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\larchCptDoc.obj" : $(SOURCE) $(DEP_CPP_LARCH) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_LARCH=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\larchCptDoc.h"\
    ".\larchCptView.h"\


"$(INTDIR)\larchCptDoc.obj" : $(SOURCE) $(DEP_CPP_LARCH) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\larchCptDoc.sbr" : $(SOURCE) $(DEP_CPP_LARCH) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\larch.rc
DEP_RSC_LARCH_=\
    ".\res\larch.ico"\
```

```
    ".\res\larchCptDoc.ico"\
    ".\res\Toolbar.bmp"\
    ".\res\cursor1.cur"\
    ".\res\larch.rc2"\


"$(INTDIR)\larch.res" : $(SOURCE) $(DEP_RSC_LARCH_) "$(INTDIR)"
   $(RSC) $(RSC_PROJ) $(SOURCE)


# End Source File
################################################################################
# Begin Source File

SOURCE=.\hlp\larch.hpj

!IF  "$(CFG)" == "larch - Win32 Release"

# Begin Custom Build - Making help file...
OutDir=.\Release
ProjDir=.
TargetName=larch
InputPath=.\hlp\larch.hpj

"$(OutDir)\$(TargetName).hlp" : $(SOURCE) "$(INTDIR)" "$(OUTDIR)"
   "$(ProjDir)\makehelp.bat"

# End Custom Build

!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

# Begin Custom Build - Making help file...
OutDir=.\Debug
ProjDir=.
TargetName=larch
InputPath=.\hlp\larch.hpj

"$(OutDir)\$(TargetName).hlp" : $(SOURCE) "$(INTDIR)" "$(OUTDIR)"
   "$(ProjDir)\makehelp.bat"

# End Custom Build

!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Serial.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_SERIA=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\
```

```
"$(INTDIR)\Serial.obj" : $(SOURCE) $(DEP_CPP_SERIA) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_SERIA=\
    ".\StdAfx.h"\
    ".\larch.h"\


"$(INTDIR)\Serial.obj" : $(SOURCE) $(DEP_CPP_SERIA) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\Serial.sbr" : $(SOURCE) $(DEP_CPP_SERIA) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\larchCptView.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_LARCHC=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\larchCptView.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Compl.h"\
    ".\larchCptDoc.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\larchCptView.obj" : $(SOURCE) $(DEP_CPP_LARCHC) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_LARCHC=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\larchCptView.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Compl.h"\
    ".\larchCptDoc.h"\


"$(INTDIR)\larchCptView.obj" : $(SOURCE) $(DEP_CPP_LARCHC) "$(INTDIR)"\
```

```
 "$(INTDIR)\larch.pch"

"$(INTDIR)\larchCptView.sbr" : $(SOURCE) $(DEP_CPP_LARCHC) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\UnComp.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_UNCOM=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\UnComp.obj" : $(SOURCE) $(DEP_CPP_UNCOM) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_UNCOM=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\


"$(INTDIR)\UnComp.obj" : $(SOURCE) $(DEP_CPP_UNCOM) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\UnComp.sbr" : $(SOURCE) $(DEP_CPP_UNCOM) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\larch.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_LARCH_C=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\MainFrm.h"\
    ".\ChildFrm.h"\
```

```
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\larchCptDoc.h"\
    ".\larchCptView.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\larch.obj" : $(SOURCE) $(DEP_CPP_LARCH_C) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_LARCH_C=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\MainFrm.h"\
    ".\ChildFrm.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\larchCptDoc.h"\
    ".\larchCptView.h"\


"$(INTDIR)\larch.obj" : $(SOURCE) $(DEP_CPP_LARCH_C) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\larch.sbr" : $(SOURCE) $(DEP_CPP_LARCH_C) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\comp1.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_COMP1=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\comp1.obj" : $(SOURCE) $(DEP_CPP_COMP1) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"
```

```
!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_COMP1=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\


"$(INTDIR)\comp1.obj" : $(SOURCE) $(DEP_CPP_COMP1) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\comp1.sbr" : $(SOURCE) $(DEP_CPP_COMP1) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\CamInt.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_CAMIN=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\larchCptDoc.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\CamInt.obj" : $(SOURCE) $(DEP_CPP_CAMIN) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_CAMIN=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\Comp1.h"\
    ".\larchCptDoc.h"\


"$(INTDIR)\CamInt.obj" : $(SOURCE) $(DEP_CPP_CAMIN) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\CamInt.sbr" : $(SOURCE) $(DEP_CPP_CAMIN) "$(INTDIR)"\
```

```
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\GetParams.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_GETPA=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\


"$(INTDIR)\GetParams.obj" : $(SOURCE) $(DEP_CPP_GETPA) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_GETPA=\
    ".\StdAfx.h"\
    ".\larch.h"\


"$(INTDIR)\GetParams.obj" : $(SOURCE) $(DEP_CPP_GETPA) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\GetParams.sbr" : $(SOURCE) $(DEP_CPP_GETPA) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
################################################################################
# Begin Source File

SOURCE=.\Comp.cpp

!IF  "$(CFG)" == "larch - Win32 Release"

DEP_CPP_COMP_=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\
    ".\GetParams.h"\
    ".\CamInt.h"\
    ".\Serial.h"\
```

```
"$(INTDIR)\Comp.obj" : $(SOURCE) $(DEP_CPP_COMP_) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ELSEIF  "$(CFG)" == "larch - Win32 Debug"

DEP_CPP_COMP_=\
    ".\StdAfx.h"\
    ".\larch.h"\
    ".\UnComp.h"\
    ".\Comp.h"\


"$(INTDIR)\Comp.obj" : $(SOURCE) $(DEP_CPP_COMP_) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"

"$(INTDIR)\Comp.sbr" : $(SOURCE) $(DEP_CPP_COMP_) "$(INTDIR)"\
 "$(INTDIR)\larch.pch"


!ENDIF

# End Source File
# End Target
# End Project
################################################################################
```

B.3.2 Resources

Since Windows is a graphical user environment, graphics and functional relationships between on-screen objects are described in the resource files.

B.3.2.1 resource.h

Resource.h contains all the message variables and object IDs for the application.

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by larch.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_CAPTURTYPE                  129
#define IDD_CAPTURE_PARAMS              130
#define IDC_PROBE                       131
#define ID_STATUS_CAPTURE               200
#define ID_STATUS_VALUE                 201
#define IDC_EDIT1                       1000
#define IDC_SPIN1                       1001
#define IDC_EDIT2                       1002
#define IDC_SPIN2                       1003
```

```
#define IDC_EDIT3                       1004
#define IDC_SPIN3                       1005
#define IDC_EDIT4                       1006
#define IDC_SPIN4                       1007
#define IDC_EDIT5                       1008
#define IDC_SPIN5                       1009
#define IDC_EDIT6                       1010
#define IDC_SPIN6                       1011
#define IDC_EDIT7                       1012
#define IDC_SPIN7                       1013
#define IDC_EDIT8                       1014
#define IDC_SPIN8                       1015
#define IDC_TEXT1                       1016
#define IDC_TEXT2                       1017
#define IDC_TEXT3                       1018
#define IDC_TEXT4                       1019
#define IDC_TEXT5                       1020
#define IDC_TEXT6                       1021
#define IDC_TEXT7                       1022
#define IDC_TEXT8                       1023
#define IDC_TITLE                       1024
#define ID_CAPTURE_START                32771
#define ID_CAPTURE_STOP                 32772
#define ID_KEY_PAGEDOWN                 32774
#define ID_KEY_PAGEUP                   32775
#define ID_CAMERA_RELOADFIRMWARE        32776
#define ID_CAMERA_LOADFPGA              32777
#define ID_KEY_DOWN                     32778
#define ID_KEY_UP                       32779
#define ID_KEY_HOME                     32780
#define ID_KEY_END                      32781
#define ID_KEY_RIGHT                    32782
#define ID_KEY_LEFT                     32783
#define ID_CAMERA_CAPTUREFILE           32784
#define ID_CAPTURE_SCROLL               32785
#define ID_CAPTURE_PARAMETERS           32786

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        132
#define _APS_NEXT_COMMAND_VALUE         32793
#define _APS_NEXT_CONTROL_VALUE         1025
#define _APS_NEXT_SYMED_VALUE           103
#endif
#endif
```

B.3.2.2 larch.rc

Larch.rc contains information on the various menus, dialog boxes with some references to on-screen bitmap files.

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif\r\n"
    "#include ""res\\larch.rc2""  // non-Microsoft Visual C++ edited
resources\r\n"
    "#include ""afxres.rc""          // Standard components\r\n"
    "#include ""afxprint.rc""       // printing/print preview resources\r\n"
    "#endif\0"
END
```

```
#endif    // APSTUDIO_INVOKED


//////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\larch.ico"
IDR_CAPTURTYPE          ICON    DISCARDABLE     "res\\larchCptDoc.ico"

//////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\Toolbar.bmp"

//////////////////////////////////////////////////////////////////////////////
//
// Toolbar
//

IDR_MAINFRAME TOOLBAR DISCARDABLE  16, 15
BEGIN
    BUTTON      ID_FILE_NEW
    BUTTON      ID_FILE_OPEN
    BUTTON      ID_FILE_SAVE
    SEPARATOR
    BUTTON      ID_EDIT_CUT
    BUTTON      ID_EDIT_COPY
    BUTTON      ID_EDIT_PASTE
    SEPARATOR
    BUTTON      ID_CAPTURE_START
    BUTTON      ID_CAPTURE_STOP
    BUTTON      ID_CAPTURE_SCROLL
    BUTTON      ID_CAPTURE_PARAMETERS
    SEPARATOR
    BUTTON      ID_FILE_PRINT
    BUTTON      ID_APP_ABOUT
    BUTTON      ID_CONTEXT_HELP
END


//////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
```

```
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Camera"
    BEGIN
        MENUITEM "&Reload Firmware",            ID_CAMERA_RELOADFIRMWARE
        MENUITEM "&Load FPGA Bitmap...",        ID_CAMERA_LOADFPGA
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help Topics",                ID_HELP_FINDER, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&About Larch Capture...",     ID_APP_ABOUT
    END
END

IDR_CAPTURTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM "&Close",                      ID_FILE_CLOSE
        MENUITEM "&Save\tCtrl+S",               ID_FILE_SAVE
        MENUITEM "Save &As...",                 ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "&Print...\tCtrl+P",           ID_FILE_PRINT
        MENUITEM "Print Pre&view",              ID_FILE_PRINT_PREVIEW
        MENUITEM "P&rint Setup...",             ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Capture"
    BEGIN
        MENUITEM "&Start",                      ID_CAPTURE_START
        MENUITEM "S&top",                       ID_CAPTURE_STOP
        MENUITEM "Scroll to &bottom",           ID_CAPTURE_SCROLL
        MENUITEM "Capture Parameters...",       ID_CAPTURE_PARAMETERS
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                ID_EDIT_CUT
        MENUITEM "&Copy All\tCtrl+C",           ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",              ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
```

```
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window",                 ID_WINDOW_NEW
        MENUITEM "&Cascade",                    ID_WINDOW_CASCADE
        MENUITEM "&Tile",                       ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",              ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help Topics",                ID_HELP_FINDER, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&About Larch Capture...",     ID_APP_ABOUT
    END
END


////////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL, NOINVERT
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL, NOINVERT
    "P",            ID_FILE_PRINT,          VIRTKEY, CONTROL, NOINVERT
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL, NOINVERT
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL, NOINVERT
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT, NOINVERT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT, NOINVERT
    VK_DOWN,        ID_KEY_DOWN,            VIRTKEY, NOINVERT
    VK_END,         ID_KEY_END,             VIRTKEY, NOINVERT
    VK_F1,          ID_HELP,                VIRTKEY, NOINVERT
    VK_F1,          ID_CONTEXT_HELP,        VIRTKEY, SHIFT, NOINVERT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY, NOINVERT
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT, NOINVERT
    VK_HOME,        ID_KEY_HOME,            VIRTKEY, NOINVERT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT, NOINVERT
    VK_LEFT,        ID_KEY_LEFT,            VIRTKEY, NOINVERT
    VK_NEXT,        ID_KEY_PAGEDOWN,        VIRTKEY, NOINVERT
    VK_PRIOR,       ID_KEY_PAGEUP,          VIRTKEY, NOINVERT
    VK_RIGHT,       ID_KEY_RIGHT,           VIRTKEY, NOINVERT
    VK_UP,          ID_KEY_UP,              VIRTKEY, NOINVERT
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL, NOINVERT
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL, NOINVERT
END


////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 248, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
```

```
CAPTION "About Larch Capture"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,18,20
    LTEXT           "Larch Capture Version 1.0",IDC_STATIC,40,10,150,8
    LTEXT           "Copyright  1997\nThe University of Windsor",IDC_STATIC,
                    40,25,150,23
    DEFPUSHBUTTON   "OK",IDOK,209,7,32,14,WS_GROUP
END

IDD_CAPTURE_PARAMS DIALOGEX 0, 0, 202, 215
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Capture Parameters"
FONT 8, "MS Sans Serif"
BEGIN
    CTEXT           "Title",IDC_TITLE,5,6,190,8
    LTEXT           "Unused",IDC_TEXT1,15,24,130,8,WS_DISABLED
    EDITTEXT        IDC_EDIT1,155,21,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN1,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,21,11,14
    LTEXT           "Unused",IDC_TEXT2,15,45,135,8,WS_DISABLED
    EDITTEXT        IDC_EDIT2,155,42,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN2,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,42,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT3,15,66,130,8,WS_DISABLED
    EDITTEXT        IDC_EDIT3,155,63,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN3,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,63,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT4,15,87,130,8,WS_DISABLED
    EDITTEXT        IDC_EDIT4,155,84,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN4,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,84,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT5,15,108,135,8,WS_DISABLED
    EDITTEXT        IDC_EDIT5,155,105,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN5,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,105,11,14
    LTEXT           "Unused",IDC_TEXT6,15,129,130,8,WS_DISABLED
    EDITTEXT        IDC_EDIT6,155,126,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN6,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,126,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT7,15,150,135,8,WS_DISABLED
    EDITTEXT        IDC_EDIT7,155,146,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN7,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,146,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT8,15,170,135,8,WS_DISABLED
    EDITTEXT        IDC_EDIT8,155,168,30,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN8,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,168,11,14,WS_EX_STATICEDGE
    DEFPUSHBUTTON   "OK",IDOK,15,192,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,135,192,50,14
END
```

```
#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "University of Windsor\0"
            VALUE "FileDescription", "LARCH MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "LARCH\0"
            VALUE "LegalCopyright", "Copyright  1997\0"
            VALUE "OriginalFilename", "LARCH.EXE\0"
            VALUE "ProductName", "LARCH Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 241
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END
```

```
    IDD_CAPTURE_PARAMS, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 195
        TOPMARGIN, 7
        BOTTOMMARGIN, 208
    END
END
#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
//
// Cursor
//

IDC_PROBE               CURSOR  DISCARDABLE     "res\\cursor1.cur"

/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "Larch Capture"
    IDR_CAPTURTYPE          "\nCapture\nLarchCapture\nLarch Capture Files
(*.cpt)\n.CPT\nLarchCapture.Document\nLarch Capture Document"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "Larch Capture"
    AFX_IDS_IDLEMESSAGE     "For Help, press F1"
    AFX_IDS_HELPMODEMESSAGE "Select an object on which to get Help"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW             "Create a new document\nNew"
    ID_FILE_OPEN            "Open an existing document\nOpen"
    ID_FILE_CLOSE           "Close the active document\nClose"
    ID_FILE_SAVE            "Save the active document\nSave"
    ID_FILE_SAVE_AS         "Save the active document with a new name\nSave As"
    ID_FILE_PAGE_SETUP      "Change the printing options\nPage Setup"
    ID_FILE_PRINT_SETUP     "Change the printer and printing options\nPrint
Setup"
    ID_FILE_PRINT           "Print the active document\nPrint"
```

```
    ID_FILE_PRINT_PREVIEW   "Display full pages\nPrint Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version number and
copyright\nAbout"
    ID_APP_EXIT             "Quit the application; prompts to save
documents\nExit"
    ID_HELP_INDEX           "Opens Help\nHelp Topics"
    ID_HELP_FINDER          "List Help topics\nHelp Topics"
    ID_HELP_USING           "Display instructions about how to use help\nHelp"
    ID_CONTEXT_HELP         "Display help for clicked on buttons, menus and
windows\nHelp"
    ID_HELP                 "Display help for current task or command\nHelp"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
    ID_FILE_MRU_FILE5       "Open this document"
    ID_FILE_MRU_FILE6       "Open this document"
    ID_FILE_MRU_FILE7       "Open this document"
    ID_FILE_MRU_FILE8       "Open this document"
    ID_FILE_MRU_FILE9       "Open this document"
    ID_FILE_MRU_FILE10      "Open this document"
    ID_FILE_MRU_FILE11      "Open this document"
    ID_FILE_MRU_FILE12      "Open this document"
    ID_FILE_MRU_FILE13      "Open this document"
    ID_FILE_MRU_FILE14      "Open this document"
    ID_FILE_MRU_FILE15      "Open this document"
    ID_FILE_MRU_FILE16      "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE            "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE            "Switch back to the previous window pane\nPrevious
Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW           "Open another window for the active document\nNew
Window"
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the window\nArrange
Icons"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap\nCascade Windows"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping tiles\nTile
Windows"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping tiles\nTile
Windows"
    ID_WINDOW_SPLIT         "Split the active window into panes\nSplit"
END

STRINGTABLE DISCARDABLE
```

```
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the capture contents and put it on the
Clipboard\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the Clipboard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
    ID_EDIT_REPLACE         "Replace specific text with different text\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR         "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR      "Show or hide the status bar\nToggle StatusBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to save the
documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
    AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_PREVIEW_CLOSE   "Close print preview mode\nCancel Preview"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_CAPTURE_START        "Start capturing data\nStart Capture"
    ID_CAPTURE_STOP         "Stop capturing data\nStop Capture"
    ID_CAMERA_RELOADFIRMWARE "Reload camera firmware\nReload Firmware"
    ID_CAMERA_LOADFPGA      "Load camera FPGA bitmap file\nLoad FPGA Bitmap"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_CAMERA_CAPTUREFILE   "Capture raw data to a file\nCapture File"
    ID_CAPTURE_SCROLL       "Scroll image to bottom while capturing\nScroll to
bottom"
    ID_CAPTURE_PARAMETERS   "Sets the Capturing parameters\nCapture
```

```
Parameters..."
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_STATUS_CAPTURE       "CAPTURE"
    ID_STATUS_VALUE         "000"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\larch.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif
/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

B.3.3 MFC Files

To add in MFC support, the StdAfx files are referenced. Not only do these files include all Windows and MFC functions and variables, they enable significantly faster application compiling with the use of precompiled headers.

B.3.3.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

//#define VC_EXTRALEAN// Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
```

```
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>// MFC support for Windows 95 Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxmt.h>
```

B.3.3.2 StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//  larch.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

B.3.4 CLarchApp:CWinApp Class

B.3.4.1 larch.h

```
// larch.h : main header file for the LARCH application
//

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

#define LARCH_MAGIC "UW_LARCH"

#include "GetParams.h"
#include "CamInt.h"

/////////////////////////////////////////////////////////////////////////////
// CLarchApp:
// See larch.cpp for the implementation of this class
//

class CLarchApp : public CWinApp
{
public:
    // Added public members
    void UpdateAll();
    void StatusBarCapture(BOOL on);
    void StatusBarGrayValue(int value);
    void StatusBarText(LPSTR text);
    CamInt *camera;
public:
    CLarchApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLarchApp)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    //}}AFX_VIRTUAL
```

```
// Implementation

    //{{AFX_MSG(CLarchApp)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

extern CLarchApp theApp;

/////////////////////////////////////////////////////////////////////////////
```

B.3.4.2 larch.cpp

```
// larch.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "larch.h"

#include "MainFrm.h"
#include "ChildFrm.h"
#include "UnComp.h"
#include "Comp.h"
#include "Compl.h"
#include "larchCptDoc.h"
#include "larchCptView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLarchApp

BEGIN_MESSAGE_MAP(CLarchApp, CWinApp)
    //{{AFX_MSG_MAP(CLarchApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLarchApp construction

CLarchApp::CLarchApp()
{
    // TODO: add construction code here,
```

```
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CLarchApp object

CLarchApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CLarchApp initialization

BOOL CLarchApp::InitInstance()
{
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();// Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

    camera = new CamInt();

    // Disable MRU
    LoadStdProfileSettings(0 /* 5 */);  // Load standard INI file options
(including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_CAPTURTYPE,
        RUNTIME_CLASS(CLarchCptDoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CLarchCptView));
    AddDocTemplate(pDocTemplate);

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
```

```
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CLarchApp::OnAppAbout()
{
```

```
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CLarchApp commands

int CLarchApp::ExitInstance()
{
    // TODO: Add your specialized code here and/or call the base class
    delete camera;

    return CWinApp::ExitInstance();
}

// Update Status Bar
void CLarchApp::StatusBarText(LPSTR text)
{
    CMainFrame *wnd = (CMainFrame*)m_pMainWnd;

    wnd->StatusBarText(text);
    UpdateAll();
}

// Update Gray Value
void CLarchApp::StatusBarGrayValue(int value)
{
    CMainFrame *wnd = (CMainFrame*)m_pMainWnd;

    wnd->StatusBarGrayValue(value);
}

// Update CAPTURE Status
void CLarchApp::StatusBarCapture(BOOL on)
{
    CMainFrame *wnd = (CMainFrame*)m_pMainWnd;

    wnd->StatusBarCapture(on);
}

void CLarchApp::UpdateAll()
{
    m_pMainWnd->UpdateWindow();
}
```

B.3.5 CMainFrame Class

B.3.5.1 MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    // Added public members
```

```
    void StatusBarCapture(BOOL on);
    void StatusBarGrayValue(int value);
    void StatusBarText(LPSTR text);
    CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnUpdateFilePrintSetup(CCmdUI* pCmdUI);
    afx_msg void OnCameraCapturefile();
    afx_msg void OnUpdateCameraCapturefile(CCmdUI* pCmdUI);
    afx_msg void OnCameraLoadfpga();
    afx_msg void OnUpdateCameraLoadfpga(CCmdUI* pCmdUI);
    afx_msg void OnCameraReloadfirmware();
    afx_msg void OnUpdateCameraReloadfirmware(CCmdUI* pCmdUI);
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg void OnClose();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

B.3.5.2 MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "larch.h"

#include "MainFrm.h"

#ifdef _DEBUG
```

```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// Set EVENT id and delay value
#define TIMER_EVENT 123
#define TIMER_VALUE 1000

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT_SETUP, OnUpdateFilePrintSetup)
    ON_COMMAND(ID_CAMERA_CAPTUREFILE, OnCameraCapturefile)
    ON_UPDATE_COMMAND_UI(ID_CAMERA_CAPTUREFILE, OnUpdateCameraCapturefile)
    ON_COMMAND(ID_CAMERA_LOADFPGA, OnCameraLoadfpga)
    ON_UPDATE_COMMAND_UI(ID_CAMERA_LOADFPGA, OnUpdateCameraLoadfpga)
    ON_COMMAND(ID_CAMERA_RELOADFIRMWARE, OnCameraReloadfirmware)
    ON_UPDATE_COMMAND_UI(ID_CAMERA_RELOADFIRMWARE, OnUpdateCameraReloadfirmware)
    ON_WM_TIMER()
    ON_WM_CLOSE()
    //}}AFX_MSG_MAP
    // Global help commands
    ON_COMMAND(ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()

// Adjust status bar indicators
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_STATUS_VALUE,
    ID_STATUS_CAPTURE
/*
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
*/
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}
```

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    // Create Toolbar
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    // Create Status Bar
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    // Setup timer for reading incoming serial data
    SetTimer(TIMER_EVENT,TIMER_VALUE,NULL);

    int width;
    UINT dummy;

    // Setup Statusbar item widths and attributes
    m_wndStatusBar.GetPaneInfo( m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE),
dummy, dummy, width);
    m_wndStatusBar.SetPaneInfo( m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE),
ID_STATUS_VALUE, SBPS_NORMAL, width+2);
//  m_wndStatusBar.SetPaneText( m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE),
"0", TRUE);

    // Turn OFF CAPTURE Indicator
    StatusBarCapture(FALSE);
    // Disable any gray value output
    StatusBarGrayValue(-1);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
```

```
    // Add horizontal and vertical scroll bars
    cs.style|=WS_HSCROLL|WS_VSCROLL;

    return CMDIFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnUpdateFilePrintSetup(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disable printing
    pCmdUI->Enable(FALSE);
}

void CMainFrame::OnCameraCapturefile()
{
    // TODO: Add your command handler code here
    // Route to camera class
    theApp.camera->CaptureFile();
}

void CMainFrame::OnUpdateCameraCapturefile(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Enable FILE Menu
    pCmdUI->Enable(TRUE);
}

void CMainFrame::OnCameraLoadfpga()
{
    // TODO: Add your command handler code here

    // Look for all .FPG files
    CFileDialog dialog( TRUE, "fpg", NULL,
        OFN_HIDEREADONLY | OFN_EXPLORER | OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST,
        "FPGA Files (*.fpg)|*.fpg||", NULL );

    CString title("Load FPGA bitmap");

    dialog.m_ofn.lpstrTitle = title;
```

```
    if (dialog.DoModal()==IDOK)
    {
        theApp.UpdateAll();
        CString file;
        file=dialog.GetPathName();
        // Call camera class to perform the operation
        if (theApp.camera->LoadFPGA(file)==FALSE)
        {
            MessageBox((CString)"Could not load '"+file+"'.","Load FPGA Bitmap
Error",MB_ICONSTOP);
        }
    }

}

void CMainFrame::OnUpdateCameraLoadfpga(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Enable FPGA bitmap loading
    pCmdUI->Enable(TRUE);
}

void CMainFrame::OnCameraReloadfirmware()
{
    // TODO: Add your command handler code here
    // Route to camera class
    theApp.camera->ReloadFirmware();
}

void CMainFrame::OnUpdateCameraReloadfirmware(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Enable menu item
    pCmdUI->Enable(TRUE);
}

void CMainFrame::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default

    // When the timer event occurs, route to camera class
    if (nIDEvent==TIMER_EVENT)
    {
        theApp.camera->TimerHit();
    }

    CMDIFrameWnd::OnTimer(nIDEvent);
}


void CMainFrame::OnClose()
{
    // TODO: Add your message handler code here and/or call default
    // Stop timer when application closes
    KillTimer(TIMER_EVENT);

    CMDIFrameWnd::OnClose();
}
```

```
void CMainFrame::StatusBarText(LPSTR text)
{
    // Update status bar text and force redraw
    m_wndStatusBar.SetPaneText(0,text,TRUE);
    m_wndStatusBar.UpdateWindow();
}

void CMainFrame::StatusBarGrayValue(int value)
{
    // Update gray value text if between 0 and 255
    if (value<0 || value > 255)
    {
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE), SBPS_DISABLED);
    }
    else
    {
        char string[5];
        _itoa(value,string,10);
        m_wndStatusBar.SetPaneText(
m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE), string, TRUE);
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE), SBPS_NORMAL);
    }
}

void CMainFrame::StatusBarCapture(BOOL on)
{
    // Enable/Disable CAPTURE indicator
    if (on==TRUE)
    {
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_CAPTURE), SBPS_NORMAL);
    }
    else
    {
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_CAPTURE), SBPS_DISABLED);
    }
}
```

B.3.6 CChildFrame Class

B.3.6.1 ChildFrm.h

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

// Attributes
public:
```

```
// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CChildFrame)
    afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

B.3.6.2 ChildFrm.cpp

```
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "larch.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
    ON_WM_SETCURSOR()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
```

```
{
    // TODO: add member initialization code here

}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CMDIChildWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers

BOOL CChildFrame::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    // TODO: Add your message handler code here and/or call default
    // Disable gray value when pointer is off the window
    if (nHitTest != HTCLIENT)
    {
        theApp.StatusBarGrayValue(-1);
    }
    return CMDIChildWnd::OnSetCursor(pWnd, nHitTest, message);
}
```

B.3.7 CLarchCptView Class

B.3.7.1 LarchCptView.h

```
// larchCptView.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView view

class CLarchCptView : public CScrollView
```

```
{
protected:
    CLarchCptView();           // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(CLarchCptView)

// Attributes
public:
    // Add public members
    BOOL m_movetobottom;
// Operations
public:
    void MoveToBottom();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLarchCptView)
    protected:
    virtual void OnDraw(CDC* pDC);      // overridden to draw this view
    virtual void OnInitialUpdate();     // first time after construct
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CLarchCptView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
    //{{AFX_MSG(CLarchCptView)
    afx_msg void OnUpdateFilePrint(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFilePrintPreview(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFilePrintSetup(CCmdUI* pCmdUI);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    afx_msg void OnKeyPageup();
    afx_msg void OnKeyPagedown();
    afx_msg void OnKeyDown();
    afx_msg void OnKeyUp();
    afx_msg void OnKeyEnd();
    afx_msg void OnKeyHome();
    afx_msg void OnKeyLeft();
    afx_msg void OnKeyRight();
    afx_msg void OnCaptureStart();
    afx_msg void OnCaptureStop();
    afx_msg void OnUpdateCaptureStart(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCaptureStop(CCmdUI* pCmdUI);
    afx_msg void OnCaptureScroll();
    afx_msg void OnUpdateCaptureScroll(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCaptureParameters(CCmdUI* pCmdUI);
    afx_msg void OnCaptureParameters();
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnUpdateFileSave(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFileSaveAs(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFileOpen(CCmdUI* pCmdUI);
    afx_msg void OnEditCopy();
    afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
```

```
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

B.3.7.2 LarchCptView.cpp

```
// larchCptView.cpp : implementation file
//

#include "stdafx.h"
#include "larch.h"
#include "larchCptView.h"
#include "UnComp.h"
#include "Comp.h"
#include "Compl.h"
#include "larchCptDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView

IMPLEMENT_DYNCREATE(CLarchCptView, CScrollView)

CLarchCptView::CLarchCptView()
{
}

CLarchCptView::-CLarchCptView()
{
}


BEGIN_MESSAGE_MAP(CLarchCptView, CScrollView)
    //{{AFX_MSG_MAP(CLarchCptView)
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT, OnUpdateFilePrint)
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT_PREVIEW, OnUpdateFilePrintPreview)
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT_SETUP, OnUpdateFilePrintSetup)
    ON_WM_ERASEBKGND()
    ON_COMMAND(ID_KEY_PAGEUP, OnKeyPageup)
    ON_COMMAND(ID_KEY_PAGEDOWN, OnKeyPagedown)
    ON_COMMAND(ID_KEY_DOWN, OnKeyDown)
    ON_COMMAND(ID_KEY_UP, OnKeyUp)
    ON_COMMAND(ID_KEY_END, OnKeyEnd)
    ON_COMMAND(ID_KEY_HOME, OnKeyHome)
    ON_COMMAND(ID_KEY_LEFT, OnKeyLeft)
    ON_COMMAND(ID_KEY_RIGHT, OnKeyRight)
    ON_COMMAND(ID_CAPTURE_START, OnCaptureStart)
    ON_COMMAND(ID_CAPTURE_STOP, OnCaptureStop)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_START, OnUpdateCaptureStart)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_STOP, OnUpdateCaptureStop)
    ON_COMMAND(ID_CAPTURE_SCROLL, OnCaptureScroll)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_SCROLL, OnUpdateCaptureScroll)
```

```
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_PARAMETERS, OnUpdateCaptureParameters)
    ON_COMMAND(ID_CAPTURE_PARAMETERS, OnCaptureParameters)
    ON_WM_MOUSEMOVE()
    ON_WM_SETFOCUS()
    ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
    ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_AS, OnUpdateFileSaveAs)
    ON_UPDATE_COMMAND_UI(ID_FILE_OPEN, OnUpdateFileOpen)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView drawing

void CLarchCptView::OnInitialUpdate()
{
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    // always disable capture follow
    m_movetobottom=FALSE;

    CScrollView::OnInitialUpdate();

    // TODO: calculate the total size of this view
    // set scroll bar sizes based on the associated document size
    SetScrollSizes(MM_TEXT, pDoc->GetScrollSize());
}

void CLarchCptView::OnDraw(CDC* pDC)
{
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();
    // call uncompressed data class to repaint
    pDoc->m_uncompdata->Paint(pDC);

}

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView diagnostics

#ifdef _DEBUG
void CLarchCptView::AssertValid() const
{
    CScrollView::AssertValid();
}

void CLarchCptView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView message handlers

void CLarchCptView::OnUpdateFilePrint(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disbable printing
```

```
    pCmdUI->Enable(FALSE);

}

void CLarchCptView::OnUpdateFilePrintPreview(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disable print preview
    pCmdUI->Enable(FALSE);

}

void CLarchCptView::OnUpdateFilePrintSetup(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disable Print setup
    pCmdUI->Enable(FALSE);

}

BOOL CLarchCptView::OnEraseBkgnd(CDC* pDC)
{
    // TODO: Add your message handler code here and/or call default

    // Erase the background of the window with default window colour
    //  don't proceed to standard call
    CBrush br( GetSysColor( COLOR_WINDOW ) );
    FillOutsideRect( pDC, &br );

    return TRUE;                    // Erased
}

void CLarchCptView::OnKeyPageup()
{
    // TODO: Add your command handler code here
    // Move display pointer up client height for page up
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y-=rect.Height();
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyPagedown()
{
    // TODO: Add your command handler code here
    // Move display pointer down client height for page up
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y+=rect.Height();
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyDown()
```

```
{
    // TODO: Add your command handler code here
    // Move display pointer down a 10th of the client height for down
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y+=rect.Height()/10;
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyUp()
{
    // TODO: Add your command handler code here
    // Move display pointer up a 10th of the client height for up
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y-=rect.Height()/10;
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyEnd()
{
    // TODO: Add your command handler code here
    // Move to bottom for end
    CSize size=GetTotalSize();
    CPoint point(0,size.cy);
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyHome()
{
    // TODO: Add your command handler code here
    // Move to top for home
    CPoint point(0,0);
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyLeft()
{
    // TODO: Add your command handler code here
    // Move display pointer left a 10th of the client width for left
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.x-=rect.Width()/10;
    ScrollToPosition(point);
}

void CLarchCptView::OnKeyRight()
{
    // TODO: Add your command handler code here
    // Move display pointer right a 10th of the client width for right
```

```
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.x+=rect.Width()/10;
    ScrollToPosition(point);
}

void CLarchCptView::OnCaptureStart()
{
    // TODO: Add your command handler code here
    // Call camera class to start capture
    theApp.camera->StartCapture(GetDocument());
    // Disable capture button if capture started
    theApp.StatusBarCapture(theApp.camera->IsCapturing(GetDocument()));
}

void CLarchCptView::OnCaptureStop()
{
    // TODO: Add your command handler code here
    // Call camera class to start capture
    theApp.camera->StopCapture(GetDocument());
    // Enable capture button if capture stopped
    theApp.StatusBarCapture(theApp.camera->IsCapturing(GetDocument()));
}

void CLarchCptView::OnUpdateCaptureStart(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Get document info to update Capture start button indicator
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    pCmdUI->Enable(theApp.camera->CanStartCapture(pDoc,pDoc-
>m_versionmajor,pDoc->m_versionminor));
}

void CLarchCptView::OnUpdateCaptureStop(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Get document info to update Capture stop button indicator
    pCmdUI->Enable(theApp.camera->CanStopCapture(GetDocument()));
}

void CLarchCptView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    // TODO: Add your specialized code here and/or call the base class

    // Adjust scroll bar sizes on update
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    SetScrollSizes(MM_TEXT, pDoc->GetScrollSize());
}

void CLarchCptView::MoveToBottom()
{
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    // Adjust scroll bar sizes
```

```
    SetScrollSizes(MM_TEXT, pDoc->GetScrollSize());

    // Get total sizes
    CSize size=GetTotalSize();
    CRect rect;
    GetClientRect(&rect);
    // Clear area client doesn't fill
    if (rect.Height()>size.cy)
    {
        Invalidate(FALSE);
    }

    if (m_movetobottom==TRUE && rect.Height()<size.cy)
    {
        // Move scroll position to maximum bottom
        CPoint point = GetScrollPosition();
        point.y = size.cy;

        ScrollToPosition(point);
    }
}

void CLarchCptView::OnCaptureScroll()
{
    // TODO: Add your command handler code here
    // Toggle follow mode when button pressed
    m_movetobottom = !m_movetobottom;
}

void CLarchCptView::OnUpdateCaptureScroll(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Enable follow button
    pCmdUI->Enable(TRUE);
    // Set button state based
    pCmdUI->SetCheck(m_movetobottom);
}

void CLarchCptView::OnUpdateCaptureParameters(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here

    // Enable parameters button if we can capture
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    pCmdUI->Enable(theApp.camera->CanCaptureParams(pDoc->m_versionmajor,pDoc-
>m_versionminor));
}

void CLarchCptView::OnCaptureParameters()
{
    // TODO: Add your command handler code here
    // Route to camera class
    theApp.camera->CaptureParams();
}

void CLarchCptView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
```

```
    // Get scroll bar position to get a relative location
    CPoint scroll =GetScrollPosition();

    scroll = scroll + point;

//  TRACE2("Point: %d, %d\n",scroll.x,scroll.y);

    // Make sure the point exists in the document
    CSize size=GetTotalSize();

    if (scroll.x <= size.cx && scroll.y <= size.cy)
    {
        // Get gray value from document
        CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();
        int value = pDoc->m_uncompdata->GetValue(scroll.x,scroll.y);
        // Update indicator display
        theApp.StatusBarGrayValue(value);
    }
    else
    {
        // Don't show it if it doesn't exist
        theApp.StatusBarGrayValue(-1);
    }

    // Continue to MFC call
    CScrollView::OnMouseMove(nFlags, point);
}

void CLarchCptView::OnSetFocus(CWnd* pOldWnd)
{
    CScrollView::OnSetFocus(pOldWnd);

    // TODO: Add your message handler code here
    // Update capture indicator when focus changes between sub documents
    theApp.StatusBarCapture(theApp.camera->IsCapturing(GetDocument()));
}

void CLarchCptView::OnUpdateFileOpen(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disbale open function when capturing
    pCmdUI->Enable(theApp.camera->IsCapturing(NULL));
}

void CLarchCptView::OnUpdateFileSave(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disable save function when capturing
    pCmdUI->Enable(theApp.camera->IsCapturing(NULL));
}

void CLarchCptView::OnUpdateFileSaveAs(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    // Disable save as when capturing
    pCmdUI->Enable(theApp.camera->IsCapturing(NULL));
}

void CLarchCptView::OnEditCopy()
```

```
{
    // TODO: Add your command handler code here
    CDC *pDC;

    // Copy capture data to windows clipboard

    // Try to open clipboard
    if (OpenClipboard())
    {
        // Clean it first
        EmptyClipboard();

        // get a draw context
        pDC = GetDC();

        // locate our parent document
        CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

        // Call sub class
        pDoc->m_uncompdata->CopyAll(pDC);

        // Release draw context
        ReleaseDC(pDC);

        // Close clipboard
        CloseClipboard();
    }
    else
    {
        // Beep if we can't open the clipboard
        MessageBeep(0);
    }
}

void CLarchCptView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here

    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();
    // Update copy all button if sub class says we can
    pCmdUI->Enable(pDoc->m_uncompdata->CanCopyAll());

}
```

B.3.8 CLarchCptDoc Class

B.3.8.1 LarchCptDoc.h

```
// larchCptDoc.h : interface of the CLarchCptDoc class
//
/////////////////////////////////////////////////////////////////////////////

class CLarchCptDoc : public CDocument
{
protected: // create from serialization only
    CLarchCptDoc();
    DECLARE_DYNCREATE(CLarchCptDoc)
```

```
// Attributes
public:
    // Added public members
    UnComp *m_uncompdata;
    Comp *m_compdata;
    BYTE m_compressor;
    int m_linelength;
    CString m_compressorname;
    int m_versionmajor;
    int m_versionminor;
    CString m_oldtitle,m_newtitle;

// Operations
public:
    void UpdateForCapture();
    CSize GetScrollSize();
    BOOL InitDocument();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLarchCptDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void OnCloseDocument();
    virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
    virtual void SetTitle(LPCTSTR lpszTitle);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CLarchCptDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CLarchCptDoc)
    afx_msg void OnCaptureStart();
    afx_msg void OnCaptureStop();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// File magic information
#define SERIALIZE_MAGIC"CAPT"
#define SERIALIZE_BESTVERSION0x00000001

/////////////////////////////////////////////////////////////////////////////
```

B.3.8.2 LarchCptDoc.cpp

```
// larchCptDoc.cpp : implementation of the CLarchCptDoc class
//
```

```
#include "stdafx.h"
#include "larch.h"

#include "UnComp.h"
#include "Comp.h"
#include "Compl.h"
#include "larchCptDoc.h"
#include "larchCptView.h"

extern CLarchApp theApp;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc

IMPLEMENT_DYNCREATE(CLarchCptDoc, CDocument)

BEGIN_MESSAGE_MAP(CLarchCptDoc, CDocument)
    //{{AFX_MSG_MAP(CLarchCptDoc)
    ON_COMMAND(ID_CAPTURE_START, OnCaptureStart)
    ON_COMMAND(ID_CAPTURE_STOP, OnCaptureStop)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc construction/destruction

CLarchCptDoc::CLarchCptDoc()
{
    // TODO: add one-time construction code here
    // Initialize document variables
    m_compdata = NULL;
    m_uncompdata = NULL;
    m_compressor = 0;
    m_compressorname = "";
    m_versionmajor = 0;
    m_versionminor = 0;
    m_oldtitle = "";
}

CLarchCptDoc::~CLarchCptDoc()
{
}

BOOL CLarchCptDoc::OnNewDocument()
{
    m_compdata = NULL;
    m_uncompdata = NULL;

    // Try to create a new document
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
```

```
    // If we can't talk to the camera, don't create a new document
    if (theApp.camera->DocumentOpen()==FALSE)
    {
        return FALSE;
    }

    // Get information from camera class
    m_compressor = theApp.camera->GetCompressor();
    m_linelength = theApp.camera->GetLineLength();
    m_versionmajor = theApp.camera->GetVersionMajor();
    m_versionminor = theApp.camera->GetVersionMinor();
    m_compressorname = theApp.camera->GetTitle();

    if (InitDocument()==FALSE)
    {
        return FALSE;
    }

    // Test document data for testing
/*
    int i,j;
    UnCompLine *line;

    for (j=0;j<256;j++)
    {
        line=new UnCompLine(m_uncompdata,j);

        for (i=0;i<1024;i++)
            line->m_pointer[i]=0; // (i+j) & 255;

        delete line;
    }
*/

    // (SDI documents will reuse this document)

    return TRUE;
}

BOOL CLarchCptDoc::InitDocument()
{
    TRACE0("CLarchCptDoc::InitDocument()\n");

    // Create uncompressed video storage
    if (m_linelength > 0 )
    {
        m_uncompdata = new UnComp(m_linelength);
    }
    else
    {
        // Error out if it can't
        m_uncompdata = NULL;
        m_compdata = NULL;

        return FALSE;
    }

    // If data is RLE encoded, create "Comp1" RLE compressed data
```

```
    if (m_compressor==1)
    {
        m_compdata = new Compl(m_uncompdata);
    }
    else
    {
        delete m_uncompdata;
        m_uncompdata = NULL;

        return FALSE;
    }

    // Set document title to name plus compressor name
    m_newtitle = m_oldtitle + " (" + m_compressorname + ")";
    CDocument::SetTitle(m_newtitle);

    return TRUE;
}

////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc serialization

void CLarchCptDoc::Serialize(CArchive& ar)
{
    // File operations
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        // Saving data

        // Write global magic data
        ar.Write(LARCH_MAGIC,sizeof(LARCH_MAGIC)-1);
        // Write module magic data
        ar.Write(SERIALIZE_MAGIC,sizeof(SERIALIZE_MAGIC)-1);
        // Write version
        ar << (DWORD) SERIALIZE_BESTVERSION;

        // Write document information
        ar << m_compressor;
        ar << m_linelength;
        ar << m_compressorname;
        ar << m_versionmajor;
        ar << m_versionminor;

        // Call compressed data class to write data
        m_compdata->Serialize(ar);

    }
    else
    {
        // TODO: add loading code here
        // Load compressed data
        DWORD file_version;

        BYTE check_main[sizeof(LARCH_MAGIC)-1];
        BYTE check_sub[sizeof(SERIALIZE_MAGIC)-1];

        // Check global type
        ar.Read(check_main,sizeof(check_main));
```

```
        if (strncmp((const char *)check_main,LARCH_MAGIC,sizeof(check_main))!=0)
        {
            AfxThrowArchiveException(CArchiveException::badIndex);
        }

        // Check module type
        ar.Read(check_sub,sizeof(check_sub));
        if (strncmp((const char
*)check_sub,SERIALIZE_MAGIC,sizeof(check_sub))!=0)
        {
            AfxThrowArchiveException(CArchiveException::badIndex);
        }

        // Check version
        ar >> (DWORD) file_version;
        if (file_version>SERIALIZE_BESTVERSION)
        {
            AfxThrowArchiveException(CArchiveException::badSchema);
        }

        // Load document variables
        ar >> m_compressor;
        ar >> m_linelength;
        ar >> m_compressorname;
        ar >> m_versionmajor;
        ar >> m_versionminor;

        // Can we create a document?
        if (InitDocument()==FALSE)
        {
            AfxThrowArchiveException(CArchiveException::generic);
        }

        // Call compressor to finish the load
        m_compdata->Serialize(ar);

    }

}

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc diagnostics

#ifdef _DEBUG
void CLarchCptDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CLarchCptDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc commands

void CLarchCptDoc::OnCaptureStart()
```

```
{
    // TODO: Add your command handler code here
}

void CLarchCptDoc::OnCaptureStop()
{
    // TODO: Add your command handler code here
}

void CLarchCptDoc::OnCloseDocument()
{
    // TODO: Add your specialized code here and/or call the base class
    // Unallocate all memory for the document
    theApp.camera->DocumentClose(this);

    if (m_compdata != NULL)
    {
        delete m_compdata;
    }
    m_compdata = NULL;
    if (m_uncompdata != NULL)
    {
        delete m_uncompdata;
    }
    m_uncompdata = NULL;

    CDocument::OnCloseDocument();

}

BOOL CLarchCptDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    // Create most data structures here
    if (theApp.camera->DocumentOpen()==FALSE)
    {
        return FALSE;
    }

    if (!CDocument::OnOpenDocument(lpszPathName))
    {
        if (m_compdata != NULL)
        {
            delete m_compdata;
        }
        m_compdata = NULL;
        if (m_uncompdata != NULL)
        {
            delete m_uncompdata;
        }
        m_uncompdata = NULL;

        return FALSE;
    }

    // TODO: Add your specialized creation code here

    return TRUE;
}
```

```
CSize CLarchCptDoc::GetScrollSize()
{
    // Get data from compressor class
    CSize sizeTotal = m_compdata->GetSize();

    return sizeTotal;
}

void CLarchCptDoc::UpdateForCapture()
{
    // Update all capture windows associated with the single document
    POSITION pos = GetFirstViewPosition();

    while (pos != NULL)
    {
        CLarchCptView* pView = (CLarchCptView*) GetNextView(pos);
        pView->MoveToBottom();
    }
}

void CLarchCptDoc::SetTitle(LPCTSTR lpszTitle)
{
    // TODO: Add your specialized code here and/or call the base class
    TRACE0("CLarchCptDoc::SetTitle()\n");

    // Modify title to add compressor name
    CString oldtitle(lpszTitle);

    if (m_oldtitle != oldtitle)
    {
        m_oldtitle = oldtitle;
        m_newtitle = m_oldtitle + " (" + m_compressorname + ")";
    }

    CDocument::SetTitle(m_newtitle);

}
```

B.3.9 CamInt Class

B.3.9.1 CamInt.h

```
// CamInt.h : header file
//

#include "Serial.h"

/////////////////////////////////////////////////////////////////////////////
// CamInt.h object

class CamInt
{
// Attributes
public:
private:
    Serial *serial;
    CDocument*capdoc;
    int m_noncapturecount;
```

```
    int m_documentsopen;
    BYTE m_params[8];
    CFile m_file;
    BYTE m_prcblock[2048];
    CGetParams *m_paramsdialog;
    BOOL m_needdefaults;
    BOOL m_capturing;

// Operations
public:
    CamInt();
    ~CamInt();
    void ReloadFirmware();
    BOOL LoadFPGA(CString filename);
    void CaptureFile();
    BOOL IsCapturing(CDocument *doc = NULL);
    BOOL StartCapture(CDocument *doc);
    BOOL StopCapture(CDocument *doc);
    BOOL CanStartCapture(CDocument *doc, int major, int minor);
    BOOL CanStopCapture(CDocument *doc);
    BOOL DocumentOpen();
    BOOL DocumentClose(CDocument *doc);
    void TimerHit();
    void CaptureParams();
    BOOL CanCaptureParams(int major, int minor);
    void GetParams(BYTE *params);
    void SetParams(BYTE *params);
    BYTE GetCompressor();
    int GetLineLength();
    int GetVersionMajor();
    int GetVersionMinor();
    CString GetTitle();
private:
    int DecodeNybble(char *data);
    int DecodeByte(char *data);
    int DecodeWord(char *data);
    BOOL MC_SendBinaryS19(LPSTR filename);
    BOOL MC_SendAsciiS19(LPSTR filename, int entry, BOOL useint);
    BOOL MC_SendFPGA(CArchive& ar);
    BOOL MC_SendProc(CArchive& ar);
    BOOL MC_GetProc();
    BOOL MC_GoToTop();
    BOOL MC_GetFPGA(LPSTR filename);
    BOOL MC_SendFPGACommand(BYTE send);
    BOOL EnterCapture();
    BOOL LeaveCapture();
    BOOL MC_CheckLevel(BYTE get);
};

// Define FPGA bitmap file magic numbers and versions
#define FPGA_MAGIC"FPGA"
#define FPGA_BESTVERSION0x00000001

/////////////////////////////////////////////////////////////////////////////
```

B.3.9.2 CamInt.cpp

```
// CamInt.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "larch.h"

#include "UnComp.h"
#include "Comp.h"
#include "Comp1.h"
#include "larchCptDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// constant definitions
// command codes for MCU code

#define MC_GOUP          0x01
#define MC_OUTPUTLEVEL   0x02
#define MC_TOPLEVEL      0x30
#define MC_SECONDLEVEL   0x31

#define MC_INITBASE      0x20
#define MC_INITBINARYS19 0x20

#define MC_INITASCIIS19  0x21
#define MC_INITLDEEFPGA  0x22
#define MC_INITLDFPGA    0x23
#define MC_INITEEDELETE  0x24
#define MC_INITLDEEPROC  0x25
#define MC_INITGETPROC   0x26

#define MC_FILELDS19     "..\\MCU\\lds19.s19"
#define MC_PROCLOC       0x46

////////////////////////////////////////////////////////////////////////////
// CamInt

CamInt::CamInt()
{
    // Create camera data
    capdoc = NULL;
    m_documentsopen = 0;
    m_needdefaults = TRUE;
    m_capturing = FALSE;
    // create serial object
    serial = new Serial();

    // Connect to camera
    serial->Connect();

    m_paramsdialog = new CGetParams(NULL,m_prcblock);
}

CamInt::~CamInt()
{
    serial->Disconnect();
```

```
    delete serial;
    delete m_paramsdialog;
}

int CamInt::DecodeNybble(char *data)
{
    // decode ascii hex value into 0-15 value
    char work=tolower(*data);
    int value;

    value=-1;
    if (work >= '0' && work <='9')
    {
        value = work - '0';
    }
    if (work >= 'a' && work <='f')
    {
        value = work - 'a' + 10;
    }

    return value;
}

int CamInt::DecodeByte(char *data)
{
    // decode 2 ascii hex values into 0-255 value
    int high,low;
    int value;

    high = DecodeNybble(data);
    low = DecodeNybble(data+1);

    if (high < 0 || low < 0 )
    {
        value = -1;
    }
    else
    {
        value = high * 16 + low;
    }

    return value;

}

int CamInt::DecodeWord(char *data)
{
    // decode 4 ascii hex values into 0-65535 value
    int high,low;
    int value;

    high = DecodeByte(data);
    low = DecodeByte(data+2);

    if (high < 0 || low < 0 )
    {
        value = -1;
    }
    else
```

```
    {
        value = high * 256 + low;
    }

    return value;

}

BOOL CamInt::MC_SendBinaryS19(LPSTR filename)
{
    // Load and S19 file and execute it in the camera MCU
    int i,j,chksum,check,size;
    int current,offset=256;
    int prgsize=0,linepoint=0;
    BYTE prghold[257];
    BYTE linehold[50];
    CFile file;
    char line[100];

    // Make sure we are at the top command level in the camera
    if (MC_GoToTop()==FALSE)
    {
        return FALSE;
    }

    // Try to open the file
    if( !file.Open( filename, CFile::modeRead | CFile::shareDenyNone) )
    {
        return FALSE;
    }

    CArchive ar( &file, CArchive::load);

    // Read in the lines and process them
    while (ar.ReadString( line, sizeof(line)-1 )!=NULL)
    {
        // If the first character of a line is not an S, error out
        if (tolower(line[0])!='s')
        {
            ar.Close();
            file.Close();
            return FALSE;
        }

        // decode size
        size = DecodeByte(line+2);
        // decode memory address
        current = DecodeWord(line+4);

        // Check sanity of file
        if (size<3 || current < 0)
        {
            ar.Close();
            file.Close();
            return FALSE;
        }

        // Start creating checksum
        chksum = (size + DecodeByte(line+4) + DecodeByte(line+6) ) & 255;
```

```
    // Read the rest of the line and store in buffer
    linepoint=8;
    for (i=0;i<size-3;i++)
    {
        j = DecodeByte(line+linepoint);
        if (j<0)
        {
            ar.Close();
            file.Close();
            return FALSE;
        }
        chksum = ( chksum + j ) & 255;
        linehold[i]=j;
        linepoint+=2;
    }

    // verify checksum
    check = DecodeByte(line+linepoint);
    if (check<0)
    {
        ar.Close();
        file.Close();
        return FALSE;
    }

    // Error out if not correct
    if (check != ( (~chksum) & 255) )
    {
        ar.Close();
        file.Close();
        return FALSE;
    }

    // Check for S1 line
    if (line[1]=='1')
    {
        if (current != offset)
        {
            ar.Close();
            file.Close();
            return FALSE;
        }
        // move buffer into transmission buffer
        for (i=0;i<size-3;i++)
        {
            prghold[prgsize]=linehold[i];
            prgsize++;
        }
        offset+=(size-3);
    }

}

// Close file
ar.Close();
file.Close();

// Send command to MCU
```

```
    serial->SendByte(MC_INITBINARYS19);
    serial->SendByte((BYTE)prgsize);

    // Send data to MCU
    for (i=0;i<prgsize;i++)
    {
        serial->SendByte((BYTE)prghold[i]);
    }

    return TRUE;
}

BOOL CamInt::MC_SendAsciiS19(LPSTR filename, int entry, BOOL useint)
{
    // Send an ASCII S19 file to MCU for it to interpret
    size_t i;
    CFile file;
    BYTE data;
    char line[100];

    // Try to open the file
    if( !file.Open( filename, CFile::modeRead | CFile::shareDenyNone ) )
    {
        return FALSE;
    }

    // Use the EEPROM filing system loader
    if (useint==TRUE)
    {
        if (MC_GoToTop()==FALSE)
        {
            file.Close();
            return FALSE;
        }
        // Send command...  Wait...
        serial->SendByte(MC_INITASCIIS19);
        Sleep(100);
    }
    else
    {
        // No?  Use binary loader and leave
        if (MC_SendBinaryS19(MC_FILELDS19)==FALSE)
        {
            file.Close();
            return FALSE;
        }
    }

    CArchive ar( &file, CArchive::load);

    // Send file number
    serial->SendByte((entry & 15) + '0');

    // Read each line
    while (ar.ReadString( line, sizeof(line)-1 )!=NULL)
    {
        // Send a line
        for(i=0;i<strlen(line);i++)
        {
```

```
            if (serial->ReceiveEmpty()==FALSE)
            {
                // See if there is an error
                if (serial->GetByte()=='E')
                {
                    ar.Close();
                    file.Close();
                    return FALSE;
                }
            }
            Sleep(2);
            serial->SendByte(line[i]);
        }
    }

    ar.Close();
    file.Close();

    // Check for level
    data=serial->GetByte();
    if (data!=MC_TOPLEVEL)
    {
        return FALSE;
    }

    return TRUE;
}

BOOL CamInt::MC_SendProc(CArchive& ar)
{
    // Send FPGA processor definition
    BYTE data;
    BYTE high,low;
    int total;

    // Go to top level
    if (MC_GoToTop()==FALSE)
    {
        return FALSE;
    }

    // Send command to delete current one
    serial->SendByte(MC_INITEEDELETE);
    // Delete appropriate file
    serial->SendByte(MC_PROCLOC);

    // Check outcome
    data=serial->GetByte();
    if (data!=MC_TOPLEVEL)
    {
        return FALSE;
    }

    // Update user on status
    theApp.StatusBarText("Transmitting Header...");

    // Send command
    serial->SendByte(MC_INITLDEEPROC);
    // Send file number
```

```
    serial->SendByte(MC_PROCLOC);

    // Send total number of bytes
    ar >> high;
    Sleep(1);
    serial->SendByte(high);
    ar >> low;
    Sleep(1);
    serial->SendByte(low);

    total = high * 256 + low;

    // Send data
    while (total > 0)
    {
        Sleep(1);
        ar >> (BYTE) data;
        serial->SendByte(data);
//      TRACE1("Send: %x\n",data);
        total--;
    }

    // Update user on status
    theApp.StatusBarText("Done.");

    // Move to top level
    data=serial->GetByte();
    if (data!=MC_TOPLEVEL)
    {
        return FALSE;
    }

    return TRUE;
}

BOOL CamInt::MC_GetProc()
{
    BYTE data;
    BYTE high,low;
    int i,total;

    // Go to top level
    if (MC_GoToTop()==FALSE)
    {
        return FALSE;
    }

    // Update status to user
    theApp.StatusBarText("Receiving Algorithm Information...");

    // Disable flow control
    serial->FlowOff();

    // Send command
    serial->SendByte(MC_INITGETPROC);
    // Send file number
    serial->SendByte(MC_PROCLOC);

    // Get size
```

```
    m_prcblock[0] = high = serial->GetByte();
    m_prcblock[1] = low = serial->GetByte();

    total = high * 256 + low;
    i=2;

    // Get data
    if (total > 0 )
    {
        while (total > 0)
        {
            m_prcblock[i] = serial->GetByte();
            total--;
            i++;
        }
        total=1;
    }

    // Goto top level
    data = serial->GetByte();
    if (data!=MC_TOPLEVEL)
    {
        serial->FlowOn();
        return FALSE;
    }

    serial->FlowOn();

    // Update status to user
    theApp.StatusBarText("Done.");

    if (total == 0)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
    }
}

// Read FPGA bitstream from file and send to camera
BOOL CamInt::MC_SendFPGA(CArchive& ar)
{
    BYTE data,low,high;
    int i,k,l=0,length;
    BYTE buffer[100];
    char percent[30];

    // Read identifier
    ar >> high;
    ar >> low;
    length = high * 256 + low;
    ar.Read(buffer,length);

    // Read version
    ar >> high;
    ar >> low;
    length = high * 256 + low;
```

```
if (length != 1 )
{
   return FALSE;
}

while (ar.Read(&data,1))
{
    // Skip Xilinx text fields
    if (data>='a' && data <='d')
    {
        ar >> high;
        ar >> low;
        length = high * 256 + low;
        ar.Read(buffer,length);
        TRACE2("CamInt::MC_SendFPGA(%x, %s)\n",data,buffer);
    }
    // Send actual data
    else if (data == 'e')
    {
        ar >> high;
        ar >> low;
        length = high * 256 + low;
        if (length != 0)
        {
            return FALSE;
        }
        ar >> high;
        ar >> low;
        length = high * 256 + low;

        k=i=length / 100;

        // Go to top level
        if (MC_GoToTop()==FALSE)
        {
            return FALSE;
        }

        // Send command
        serial->SendByte(MC_INITLDEEFPGA);

        // Read data
        while (ar.Read(&data,1))
        {
            if (k>=i)
            {
                // Update user on status
                sprintf(percent,"Transmitting: %d%%",l/i);
                theApp.StatusBarText(percent);
                k=-1;
            }
            l++;
            k++;

            // Check for error
            if (serial->ReceiveEmpty()==FALSE)
            {
                if (serial->GetByte()=='E')
                {
```

```
                    return FALSE;
                }
            }
            Sleep(2);
            serial->SendByte(data);
        }

        data=serial->GetByte();
        if (data!=MC_TOPLEVEL)
        {
            return FALSE;
        }

        // Update user on status
        theApp.StatusBarText("Transmission Successful");

    }
}


    return TRUE;
}

BOOL CamInt::MC_GoToTop()
{
    // Go to top
    int i=0;
    BYTE data;

    // Flush incoming data buffers
    serial->ClearIncoming();
    serial->ClearBuffer();

    // Try for 10 times
    while (i<10)
    {
        // Send top level command, wait...
        serial->SendByte(MC_GOUP);
        Sleep(100);
        // check incoming data
        data=serial->GetByte();
        if (data==MC_TOPLEVEL)
        {
            // Wait for it to stop sending stuff
            Sleep(100);
            while (serial->ReceiveEmpty()==FALSE)
            {
                serial->GetByte();
                Sleep(100);
            }

            return TRUE;
        }
        i++;
    }

    return FALSE;

}
```

```
BOOL CamInt::MC_GetFPGA(LPSTR filename)
{
    // Test code!!!
    int i;
    CFile file;
    BYTE data;

    // Try to openfile
    if( !file.Open( filename, CFile::modeCreate | CFile::modeWrite ) )
    {
        return FALSE;
    }

    if (MC_GoToTop()==FALSE)
    {
        return FALSE;
    }

    // Send FPGA load command
    MC_SendFPGACommand(MC_INITLDFPGA);

    // Setup capture
    MC_SendFPGACommand('B');
    MC_SendFPGACommand('C');
    MC_SendFPGACommand('A');

    serial->SendByte('D');
    serial->SendByte(m_params[0]);
    MC_SendFPGACommand(0);

    serial->SendByte('E');

    serial->FlowOff();

    // Capture 8K of data
    for (i=0;i<8192;i++)
    {
        data=serial->GetByte();
        file.Write(&data,1);
    }

    // Stop sending
    serial->SendByte(MC_GOUP);

    Sleep(1000);

    serial->FlowOn();

    serial->ClearIncoming();
    serial->ClearBuffer();

    file.Close();

    return MC_GoToTop();

}

BOOL CamInt::MC_SendFPGACommand(BYTE send)
```

```
{
    BYTE data;

    // Send a command to the capture, make sure we are in second level mode
    serial->SendByte(send);

    data=serial->GetByte();
    if (data!=MC_SECONDLEVEL)
    {
        return FALSE;
    }

    return TRUE;
}

BOOL CamInt::MC_CheckLevel(BYTE get)
{
    BYTE data;

    // Check see if current level is what we think
    serial->SendByte(MC_OUTPUTLEVEL);

    data=serial->GetByte();
    if (data!=get)
    {
        return FALSE;
    }

    return TRUE;

}

void CamInt::ReloadFirmware()
{
    // Reload firmware
    CWaitCursor wait;

    // Order of these files depends on the commands sent to the MCU
    // Update the user constantly
    theApp.StatusBarText("Transmitting EEPROM Formatter...");
    MC_SendBinaryS19("..\\MCU\\eeformat.s19");
    theApp.StatusBarText("Transmitting S19 Loader...");
    MC_SendAsciiS19("..\\MCU\\lds19.s19",MC_INITASCIIS19-MC_INITBASE,FALSE);
    theApp.StatusBarText("Transmitting FPGA Loader...");
    MC_SendAsciiS19("..\\MCU\\ldeefpga.s19",MC_INITLDEEFPGA-MC_INITBASE,TRUE);
    theApp.StatusBarText("Transmitting FPGA Executer...");
    MC_SendAsciiS19("..\\MCU\\ldfpga.s19",MC_INITLDFPGA-MC_INITBASE,TRUE);
    theApp.StatusBarText("Transmitting EEPROM Deleter...");
    MC_SendAsciiS19("..\\MCU\\eedelete.s19",MC_INITEEDELETE-MC_INITBASE,TRUE);
    theApp.StatusBarText("Transmitting PROC Loader...");
    MC_SendAsciiS19("..\\MCU\\ldeeproc.s19",MC_INITLDEEPROC-MC_INITBASE,TRUE);
    theApp.StatusBarText("Transmitting PROC Transmitter...");
    MC_SendAsciiS19("..\\MCU\\getproc.s19",MC_INITGETPROC-MC_INITBASE,TRUE);
    theApp.StatusBarText("Firmware Reloaded Successfully");
}

// Load FPG file into camera
BOOL CamInt::LoadFPGA(CString filename)
{
```

```
CWaitCursor wait;

// Load FPGA file (includes definition, and bitmap)
DWORD file_version;
CFile file;

BYTE check_main[sizeof(LARCH_MAGIC)-1];
BYTE check_sub[sizeof(FPGA_MAGIC)-1];

if( !file.Open( filename, CFile::modeRead | CFile::shareDenyNone) )
{
    return FALSE;
}

CArchive ar( &file, CArchive::load);

// Make sure file type and version are correct
ar.Read(check_main,sizeof(check_main));
if (strncmp((const char *)check_main,LARCH_MAGIC,sizeof(check_main))!=0)
{
    ar.Close();
    file.Close();
    return FALSE;
}

ar.Read(check_sub,sizeof(check_sub));
if (strncmp((const char *)check_sub,FPGA_MAGIC,sizeof(check_sub))!=0)
{
    ar.Close();
    file.Close();
    return FALSE;
}

ar >> (DWORD) file_version;
if (file_version>FPGA_BESTVERSION)
{
    ar.Close();
    file.Close();
    return FALSE;
}

// Send definition
if (MC_SendProc(ar)==FALSE)
{
    ar.Close();
    file.Close();
    return FALSE;
}

// Send FPGA bitstream
if (MC_SendFPGA(ar)==FALSE)
{
    ar.Close();
    file.Close();
    return FALSE;
}

ar.Close();
file.Close();
```

```
    m_needdefaults = TRUE;

    return TRUE;
}

void CamInt::CaptureFile()
{
    CWaitCursor wait;
    theApp.StatusBarText("Capturing...");
    MC_GetFPGA("in.dat");
    theApp.StatusBarText("Done");
}

// Open a document
BOOL CamInt::DocumentOpen()
{
    TRACE0("CamInt::DocumentOpen()\n");

    if (m_documentsopen==0)
    {
        // Get ready for capture
        if (EnterCapture()==FALSE)
        {
            return FALSE;
        }
    }

    // Count documents open
    m_documentsopen++;

    return TRUE;
}

// Close a socument
BOOL CamInt::DocumentClose(CDocument *doc)
{
    TRACE0("CamInt::DocumentClose()\n");

    // decrease open documents, keep it above zero
    m_documentsopen--;
    m_documentsopen = max(0,m_documentsopen);

    // If capturing, stop it
    if (capdoc==doc)
    {
        StopCapture(capdoc);
    }
    if (m_documentsopen==0)
    {
        // Leave capture level
        if (LeaveCapture()==FALSE)
        {
            return FALSE;
        }
    }
    return TRUE;
}
```

```
// Move to capture level inside camera
BOOL CamInt::EnterCapture()
{
    CWaitCursor wait;

    TRACE0("CamInt::EnterCapture()\n");

    if (MC_GetProc()==FALSE)
    {
        return FALSE;
    }

    if (m_needdefaults == TRUE)
    {
        m_paramsdialog->UseDefaultParams();
        m_paramsdialog->GetParams(m_params);
        m_needdefaults = FALSE;
    }

    if (MC_GoToTop()==FALSE)
    {
        return FALSE;
    }

    MC_SendFPGACommand(MC_INITLDFPGA);

    MC_SendFPGACommand('B');
    MC_SendFPGACommand('C');

    return MC_CheckLevel(MC_SECONDLEVEL);
}

// Go backto command level (stop capture first)
BOOL CamInt::LeaveCapture()
{
    CWaitCursor wait;

    TRACE0("CamInt::LeaveCapture()\n");
    if (MC_CheckLevel(MC_SECONDLEVEL)==FALSE)
    {
        return FALSE;
    }

    MC_SendFPGACommand('B');
    MC_SendFPGACommand('C');

    return MC_GoToTop();
}

// Are we capturing?
BOOL CamInt::IsCapturing(CDocument *doc)
{
    if (doc == NULL)
    {
        return !m_capturing;
    }

    if (capdoc==doc) return TRUE;
```

```
    return FALSE;
}

// Can we start capturing?
BOOL CamInt::CanStartCapture(CDocument *doc, int major, int minor)
{
    if (capdoc!=NULL) return FALSE;

    if (major != GetVersionMajor() || minor != GetVersionMinor() ) return FALSE;

    return TRUE;
}

// Can we change the parameters based on the current processor
BOOL CamInt::CanCaptureParams(int major, int minor)
{
    if (major != GetVersionMajor() || minor != GetVersionMinor() ) return FALSE;

    return TRUE;
}

// Start capturing
BOOL CamInt::StartCapture(CDocument *doc)
{
    CWaitCursor wait;

    TRACE0("CamInt::StartCapture()\n");
    if (capdoc!=NULL) return FALSE;

    MC_SendFPGACommand('B');
    MC_SendFPGACommand('C');

    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        serial->SendByte('D');
        serial->SendByte(m_params[i]);
        MC_SendFPGACommand(i);
    }

    MC_SendFPGACommand('A');

    Sleep(1);

    serial->FlowOff();

    serial->SendByte('E');

    m_noncapturecount=0;

    // Open a debugging file (won't be more than 128K)
    if( !m_file.Open( "d:\\probe.dat", CFile::modeCreate | CFile::modeWrite ) )
    {
        return FALSE;
    }

    capdoc = doc;
    m_capturing = TRUE;
```

```
    return TRUE;
}

// Can we stop capturing?
BOOL CamInt::CanStopCapture(CDocument *doc)
{
    if (capdoc!=doc) return FALSE;

    return TRUE;
}

// Stop capturing
BOOL CamInt::StopCapture(CDocument *doc)
{
    CWaitCursor wait;

    TRACE0("CamInt::StopCapture()\n");
    if (capdoc!=doc) return FALSE;

    // Flush buffer
    TimerHit();
    MC_SendFPGACommand(MC_GOUP);

    Sleep(1000);

    serial->ClearIncoming();
    serial->ClearBuffer();

    CLarchCptDoc *pdoc=(CLarchCptDoc*)capdoc;
    pdoc->m_compdata->EndData();

    serial->FlowOn();

    serial->ClearIncoming();
    serial->ClearBuffer();

    MC_SendFPGACommand('B');

    m_file.Close();
    capdoc=NULL;
    m_capturing = FALSE;

    return TRUE;
}

// Process current buffer of data
void CamInt::TimerHit()
{
    if (capdoc==NULL) return;

    CLarchCptDoc *doc=(CLarchCptDoc*)capdoc;

    BYTE temp[8192];
    int num;

    num=serial->AvailableBytes();

    TRACE1("CamInt::TimerHit() with %d bytes.\n",num);
```

```
    if (num>0)
    {
        // Process compressed data
        num=serial->GetBytes(temp,num);
        m_file.Write(temp,num);

        doc->m_compdata->AddData(temp,num);

        doc->m_compdata->ProcessData();
        doc->UpdateForCapture();
    }
    else
    {
        // If we didn't get data for 5 seconds, stop capture
        m_noncapturecount++;
        if (m_noncapturecount>5)
        {
            m_noncapturecount = 0;
            StopCapture(capdoc);
        }
    }

    return;
}

// Set processor parameters
void CamInt::CaptureParams()
{
    CFile file;

    // Set parameters
    m_paramsdialog->SetParams(m_params);
    // Do dialog
    m_paramsdialog->DoModal();
    // Get parameters
    m_paramsdialog->GetParams(m_params);

    TRACE2("Params: %d, %d, ",(int)m_params[0],(int)m_params[1]);
    TRACE2("%d, %d\n",(int)m_params[2],(int)m_params[3]);
}

// Create a copy of the parameters
void CamInt::GetParams(BYTE *params)
{
    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        params[i]=m_params[i];
    }
}

// Update parameters
void CamInt::SetParams(BYTE *params)
{
    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        m_params[i]=params[i];
    }
```

```
}

// Get compressor type
BYTE CamInt::GetCompressor()
{
    return m_paramsdialog->GetCompressor();
}

// Get line length
int CamInt::GetLineLength()
{
    return m_paramsdialog->GetLineLength();
}

// Get version major
int CamInt::GetVersionMajor()
{
    return m_paramsdialog->GetVersionMajor();
}

// Get version minor
int CamInt::GetVersionMinor()
{
    return m_paramsdialog->GetVersionMinor();
}

// Get title
CString CamInt::GetTitle()
{
    return m_paramsdialog->GetTitle();
}

////////////////////////////////////////////////////////////////////////////
```

B.3.10 Serial Class

B.3.10.1 Serial.h

```
// Serial.h : header file
//

////////////////////////////////////////////////////////////////////////////
// Serial object

class Serial
{
// Attributes
public:
    CSingleLock *lock;
    CEvent*event;
private:
    HANDLEComDev;
    BYTEByteSize, FlowCtrl, Parity, StopBits ;
    DWORDBaudRate ;
    CWinThread *SerialThread;
    BOOLConnected;
    BOOL SerialThreadRunning;
    OVERLAPPEDosWrite, osRead;
```

```
    BYTEbuffer[8192];
    int buf_read;
    int buf_write;
// Operations
public:
    Serial();
    ~Serial();
    BOOL Connect();
    Disconnect();
    void SetSerialThreadRunning(BOOL mode);
    OVERLAPPED *GetosRead();
    HANDLE GetComDev();
    BOOL GetConnected();
    BYTE GetByte();
    int GetBytes(BYTE *data, int length);
//  BYTE GetByteFlow();
    BOOL ReceiveEmpty();
    BOOL SendByte(BYTE data);
//  BOOL SendByteFlow(BYTE data);
    void AddBytes(BYTE *data, int length);
    void ClearBuffer();
    void ClearIncoming();
    BOOL SerialWriteBlock(BYTE *lpByte , DWORD dwBytesToWrite);
    void FlowOn();
    void FlowOff();
    int AvailableBytes();
private:

};

UINT SerialReceiveProc( LPVOID pParam );
int SerialReadBlock(Serial *pObject, LPSTR lpszBlock, int nMaxLength);

WINBASEAPI
BOOL
WINAPI
ClearCommBreak(
    HANDLE hFile
    );

WINBASEAPI
BOOL
WINAPI
ClearCommError(
    HANDLE hFile,
    LPDWORD lpErrors,
    LPCOMSTAT lpStat
    );

WINBASEAPI
BOOL
WINAPI
SetupComm(
    HANDLE hFile,
    DWORD dwInQueue,
    DWORD dwOutQueue
    );

WINBASEAPI
```

```
BOOL
WINAPI
EscapeCommFunction(
    HANDLE hFile,
    DWORD dwFunc
    );

WINBASEAPI
BOOL
WINAPI
GetCommConfig(
    HANDLE hCommDev,
    LPCOMMCONFIG lpCC,
    LPDWORD lpdwSize
    );

WINBASEAPI
BOOL
WINAPI
GetCommMask(
    HANDLE hFile,
    LPDWORD lpEvtMask
    );

WINBASEAPI
BOOL
WINAPI
GetCommProperties(
    HANDLE hFile,
    LPCOMMPROP lpCommProp
    );

WINBASEAPI
BOOL
WINAPI
GetCommModemStatus(
    HANDLE hFile,
    LPDWORD lpModemStat
    );

WINBASEAPI
BOOL
WINAPI
GetCommState(
    HANDLE hFile,
    LPDCB lpDCB
    );

WINBASEAPI
BOOL
WINAPI
GetCommTimeouts(
    HANDLE hFile,
    LPCOMMTIMEOUTS lpCommTimeouts
    );

WINBASEAPI
BOOL
WINAPI
```

```
PurgeComm(
    HANDLE hFile,
    DWORD dwFlags
    );

WINBASEAPI
BOOL
WINAPI
SetCommBreak(
    HANDLE hFile
    );

WINBASEAPI
BOOL
WINAPI
SetCommConfig(
    HANDLE hCommDev,
    LPCOMMCONFIG lpCC,
    DWORD dwSize
    );

WINBASEAPI
BOOL
WINAPI
SetCommMask(
    HANDLE hFile,
    DWORD dwEvtMask
    );

WINBASEAPI
BOOL
WINAPI
SetCommState(
    HANDLE hFile,
    LPDCB lpDCB
    );

WINBASEAPI
BOOL
WINAPI
SetCommTimeouts(
    HANDLE hFile,
    LPCOMMTIMEOUTS lpCommTimeouts
    );

WINBASEAPI
BOOL
WINAPI
TransmitCommChar(
    HANDLE hFile,
    char cChar
    );

WINBASEAPI
BOOL
WINAPI
WaitCommEvent(
    HANDLE hFile,
    LPDWORD lpEvtMask,
```

```
    LPOVERLAPPED lpOverlapped
    );

////////////////////////////////////////////////////////////////////////////
```

B.3.10.2 Serial.cpp

```
// Serial.cpp : implementation file
//

#include "stdafx.h"
#include "larch.h"
// #include "serial.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// constant definitions

#define PORTSECTION     "Communications"
#define PORTNAME        "SerialPort"
// COM port
#define PORTDEFAULT     "COM2"

#define RXQUEUE         4096
#define TXQUEUE         0
#define MAXBLOCK512

#define ASCII_LF        0x0A
#define ASCII_CR        0x0D
#define ASCII_XON       0x11
#define ASCII_XOFF      0x13

////////////////////////////////////////////////////////////////////////////
// Serial

// Create Serial object
Serial::Serial()
{
    BaudRate = CBR_9600; // 9600 baud
    Parity = NOPARITY; // None
    StopBits = ONESTOPBIT;  // 1 stopbit
    ByteSize = 8; // 8 bit word
    SerialThread = NULL;
    SerialThreadRunning = FALSE;
    Connected = FALSE;
    buf_read = buf_write = 0;
    TRACE0("Serial()\n");
    // Create synchronization object
    event = new CEvent(TRUE,FALSE,"SERIAL",NULL);
    event->Unlock();
    lock = new CSingleLock(event,FALSE);
}

// Destory Serial object
```

```
Serial::~Serial()
{
    if (Connected==TRUE) Disconnect();

    delete lock;
    delete event;

    TRACE0("~Serial()\n");
}

// Connect to port
BOOL Serial::Connect()
{
    COMMTIMEOUTSCommTimeOuts ;
    DCB dcb ;

    TRACE0("Connect()\n");

    CString port;

    port = theApp.GetProfileString(PORTSECTION,PORTNAME,PORTDEFAULT);

    if ((ComDev =
        CreateFile( port, GENERIC_READ | GENERIC_WRITE,
                    0,                    // exclusive access
                    NULL,                 // no security attrs
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL |
                    FILE_FLAG_OVERLAPPED, // overlapped I/O
                    NULL )) == (HANDLE) -1 )
    return FALSE;

    // get any early notifications

    SetCommMask( ComDev, EV_RXCHAR ) ;

    // setup device buffers

    SetupComm( ComDev, RXQUEUE, TXQUEUE ) ;

    // purge any information in the buffer

    PurgeComm( ComDev, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR |
PURGE_RXCLEAR ) ;

    // set up for overlapped I/O

    CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF ;
    CommTimeOuts.ReadTotalTimeoutMultiplier = 0 ;
    CommTimeOuts.ReadTotalTimeoutConstant = 1000 ;
    CommTimeOuts.WriteTotalTimeoutMultiplier = 0 ;
    CommTimeOuts.WriteTotalTimeoutConstant = 1000 ;
    SetCommTimeouts( ComDev, &CommTimeOuts ) ;

    dcb.DCBlength = sizeof( DCB ) ;

    GetCommState( ComDev, &dcb ) ;

    dcb.BaudRate = BaudRate;
```

```
dcb.ByteSize = ByteSize;
dcb.StopBits = StopBits;

dcb.fBinary = TRUE;
dcb.fNull = FALSE;
dcb.fAbortOnError = FALSE;

// Parity stuff
dcb.Parity = Parity;
dcb.fParity = FALSE;
dcb.fErrorChar = FALSE;

// Flow control stuff
dcb.fOutxCtsFlow = FALSE;
dcb.fOutxDsrFlow = FALSE;
dcb.fDtrControl = DTR_CONTROL_DISABLE;
dcb.fDsrSensitivity = FALSE;
dcb.fRtsControl = RTS_CONTROL_DISABLE;

dcb.fTXContinueOnXoff = FALSE;
dcb.fOutX = TRUE; // FALSE; // TRUE;
dcb.fInX = FALSE;
dcb.XonLim = 0;
dcb.XoffLim = 0;
dcb.XonChar = ASCII_XON ;
dcb.XoffChar = ASCII_XOFF ;

if (SetCommState( ComDev, &dcb )==FALSE)
{
    CloseHandle( ComDev ) ;
    return FALSE;
}

osWrite.Offset = 0 ;
osWrite.OffsetHigh = 0 ;
osRead.Offset = 0 ;
osRead.OffsetHigh = 0 ;

// create I/O event used for overlapped reads / writes
osRead.hEvent = CreateEvent( NULL,    // no security
                             TRUE,    // explicit reset req
                             FALSE,   // initial event reset
                             NULL ) ; // no name
if (osRead.hEvent == NULL)
{
    CloseHandle( ComDev ) ;
    return FALSE;
}

osWrite.hEvent = CreateEvent( NULL,    // no security
                              TRUE,    // explicit reset req
                              FALSE,   // initial event reset
                              NULL ) ; // no name
if (osWrite.hEvent == NULL)
{
    CloseHandle( ComDev ) ;
    CloseHandle( osRead.hEvent ) ;
    return FALSE;
}
```

```
    // Create thread to receive data
    SerialThread = AfxBeginThread(SerialReceiveProc, this);

    Connected = TRUE;

    TRACE0("Connect(TRUE)\n");

    return TRUE;
}

// Enable flow control
void Serial::FlowOn()
{
    DCB dcb ;

    dcb.DCBlength = sizeof( DCB ) ;
    GetCommState( ComDev, &dcb ) ;
    dcb.fOutX = TRUE;
    SetCommState( ComDev, &dcb );
}

// Disable flow control
void Serial::FlowOff()
{
    DCB dcb ;

    dcb.DCBlength = sizeof( DCB ) ;
    GetCommState( ComDev, &dcb ) ;
    dcb.fOutX = FALSE;
    SetCommState( ComDev, &dcb );
}

// Disconnect port
Serial::Disconnect()
{

    Connected = FALSE ;

    // disable event notification and wait for thread
    // to halt

    SetCommMask( ComDev, 0 ) ;

    // block until thread has been halted

    while(SerialThreadRunning != FALSE);
    SerialThread = NULL;

    // purge any outstanding reads/writes and close device handle

    PurgeComm( ComDev, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR |
PURGE_RXCLEAR ) ;

    CloseHandle( ComDev ) ;
    CloseHandle( osRead.hEvent ) ;
    CloseHandle( osWrite.hEvent ) ;

    TRACE0("Disconnect()\n");
```

```
    return 0;
}

// Receiver thread
UINT SerialReceiveProc( LPVOID pParam )
{
    Serial *pObject = (Serial*)pParam;
    DWORD       dwEvtMask ;
    OVERLAPPED  os ;
    int         nLength ;
    BYTE        abIn[ MAXBLOCK + 1] ;

    TRACE0("SerialReceiveProc()\n");

    pObject->SetSerialThreadRunning(TRUE);

    memset( &os, 0, sizeof( OVERLAPPED ) ) ;

    // create I/O event used for overlapped read

    os.hEvent = CreateEvent( NULL,    // no security
                             TRUE,    // explicit reset req
                             FALSE,   // initial event reset
                             NULL ) ; // no name
    if (os.hEvent == NULL)
    {
        TRACE0("SerialReceiveProc(FALSE)\n");
        return 1;
    }

    TRACE0("SerialReceiveProc(TRUE)\n");

    if (!SetCommMask( pObject->GetComDev(), EV_RXCHAR )) return 1;

    while ( pObject->GetConnected() == TRUE)
    {
        dwEvtMask = 0 ;

        WaitCommEvent( pObject->GetComDev(), &dwEvtMask, NULL );

        if ((dwEvtMask & EV_RXCHAR) == EV_RXCHAR)
        {

            do
            {
                if (nLength = SerialReadBlock( pObject, (LPSTR) abIn, MAXBLOCK ))
                {
                    abIn[nLength]='\0';
//                  TRACE1("SerialReceiveProc(Got %d bytes)\n",nLength);
//                  TRACE1("SerialReceiveProc(Bytes: %s)\n",abIn);
                    pObject->AddBytes(abIn,nLength);
                }
            }
            while ( nLength > 0 ) ;
        }
    }

    // get rid of event handle
```

```
    CloseHandle( os.hEvent ) ;

    // clear information in structure (kind of a "we're done flag")
    pObject->SetSerialThreadRunning(FALSE);

    TRACE0("SerialReceiveProc(DONE)\n");

    return 0;// thread completed successfully
}

// Add bytes to buffer area shared by main thread
void Serial::AddBytes(BYTE *data, int length)
{
    int i;

    if (length<=0) return;

    lock->Lock();

    for (i=0;i<length;i++)
    {
        buffer[buf_write]=data[i];
        buf_write = (buf_write + 1 ) & (sizeof(buffer)-1);
    }

    event->SetEvent();
    lock->Unlock();

    return;
}

// Read serial information
int SerialReadBlock(Serial *pObject, LPSTR lpszBlock, int nMaxLength)
{
    BOOL        fReadStat ;
    COMSTAT     ComStat ;
    DWORD       dwErrorFlags;
    DWORD       dwLength;
    DWORD       dwError;

    // only try to read number of bytes in queue
    ClearCommError( pObject->GetComDev(), &dwErrorFlags, &ComStat ) ;
    dwLength = min( (DWORD) nMaxLength, ComStat.cbInQue ) ;

    if (dwLength > 0)
    {
        fReadStat = ReadFile( pObject->GetComDev(), lpszBlock,
                              dwLength, &dwLength, pObject->GetosRead() ) ;
        if (!fReadStat)
        {
            if (GetLastError() == ERROR_IO_PENDING)
            {
                TRACE0("SerialReadBlock: IO Pending\n");
                // We have to wait for read to complete.
                // This function will timeout according to the
                // CommTimeOuts.ReadTotalTimeoutConstant variable
                // Every time it times out, check for port errors
                while(!GetOverlappedResult( pObject->GetComDev(),
```

```
                pObject->GetosRead(), &dwLength, TRUE ))
            {
                dwError = GetLastError();
                if(dwError == ERROR_IO_INCOMPLETE)
                    // normal result if not finished
                    continue;
                else
                {
                    // an error occurred, try to recover
                    TRACE1("SerialReadBlock Error:%x\n",dwError);
                    ClearCommError( pObject->GetComDev(), &dwErrorFlags,
&ComStat ) ;
                    if (dwErrorFlags > 0)
                    {
                        TRACE1("SerialReadBlock Error: %x\n",dwErrorFlags);
                    }
                    break;
                }

            }

        }
        else
        {
            // some other error occurred

            dwLength = 0 ;
            ClearCommError( pObject->GetComDev(), &dwErrorFlags, &ComStat ) ;
            if (dwErrorFlags > 0)
            {
                TRACE1("SerialReadBlock Error: %x\n", dwErrorFlags ) ;
            }
        }
    }
}

    return ( dwLength ) ;

}

// Wirte serial information
BOOL Serial::SerialWriteBlock(BYTE *lpByte , DWORD dwBytesToWrite)
{

    BOOL        fWriteStat ;
    DWORD       dwBytesWritten ;
    DWORD       dwErrorFlags;
    DWORD   dwError;
    COMSTAT     ComStat;

    fWriteStat = WriteFile( ComDev, lpByte, dwBytesToWrite,
                        &dwBytesWritten, &osWrite ) ;

    // Note that normally the code will not execute the following
    // because the driver caches write operations. Small I/O requests
    // (up to several thousand bytes) will normally be accepted
    // immediately and WriteFile will return true even though an
    // overlapped operation was specified
```

```
    if (!fWriteStat)
    {
        if(GetLastError() == ERROR_IO_PENDING)
        {
            // We should wait for the completion of the write operation
            // so we know if it worked or not

            // This is only one way to do this. It might be beneficial to
            // the to place the writing operation in a separate thread
            // so that blocking on completion will not negatively
            // affect the responsiveness of the UI

            // If the write takes long enough to complete, this
            // function will timeout according to the
            // CommTimeOuts.WriteTotalTimeoutConstant variable.
            // At that time we can check for errors and then wait
            // some more.

            while(!GetOverlappedResult( ComDev ,
                &osWrite, &dwBytesWritten, TRUE ))
            {
                dwError = GetLastError();
                if(dwError == ERROR_IO_INCOMPLETE)
                {
                    // normal result if not finished
                    TRACE0("SerialWriteBlock: IO Incomplete\n");
                    continue;
                }
                else
                {
                    // an error occurred, try to recover
                    TRACE1("SerialWriteBlock Error: %x\n",dwError);
                    ClearCommError( ComDev, &dwErrorFlags, &ComStat ) ;
                    if (dwErrorFlags > 0)
                    {
                        TRACE1("SerialWriteBlock Error: Flag=%x\n",dwErrorFlags);
                    }
                    break;
                }
            }
        }
        else
        {
            // some other error occurred

            ClearCommError( ComDev, &dwErrorFlags, &ComStat ) ;
            if (dwErrorFlags > 0)
            {
                TRACE1("SerialWriteBlock Error: Flag=%x\n",dwErrorFlags);
            }
            TRACE0("SerialWriteBlock Returned FALSE.\n");
            return FALSE;
        }
    }
    return TRUE;

}

void Serial::SetSerialThreadRunning(BOOL mode)
```

```
{
    SerialThreadRunning = mode;

    return;
}

HANDLE Serial::GetComDev()
{
    return ComDev;
}

BOOL Serial::GetConnected()
{
    return Connected;
}

OVERLAPPED *Serial::GetosRead()
{
    return &osRead;
}

// Get a byte from serial buffer
BYTE Serial::GetByte()
{
    BYTE value;

    while(ReceiveEmpty());

    lock->Lock();

    value = buffer[buf_read];
    buf_read = ( buf_read + 1 ) & (sizeof(buffer)-1);

    event->SetEvent();
    lock->Unlock();

    return value;

}

// Get a series of bytes from the serial buffer
int Serial::GetBytes(BYTE *data, int length)
{
    int i,num;

    num = min(AvailableBytes(),length);

    if (num<=0) return 0;

    lock->Lock();

    for (i=0;i<num;i++)
    {
        data[i]=buffer[buf_read];
        buf_read = ( buf_read + 1 ) & (sizeof(buffer)-1);
    }

    event->SetEvent();
    lock->Unlock();
```

```
    return num;
}

// Check if buffer is empty
BOOL Serial::ReceiveEmpty()
{
    return (buf_read == buf_write);
}

// Send a byte
BOOL Serial::SendByte(BYTE data)
{
//  TRACE1("SendByte: %x\n",data);
    return SerialWriteBlock(&data , 1);
}

// Clear the receive buffer
void Serial::ClearBuffer()
{
    lock->Lock();
//  TRACE0("ClearBuffer:Locked\n");

    buf_read=buf_write=0;

    event->SetEvent();
    lock->Unlock();
//  TRACE0("ClearBuffer:UnLocked\n");
}

// Purge data in kernal
void Serial::ClearIncoming()
{
    PurgeComm( ComDev, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR |
PURGE_RXCLEAR ) ;
}

// How many bytes are available
int Serial::AvailableBytes()
{
    int num;

    lock->Lock();

    if (buf_read <= buf_write)
    {
        num = buf_write - buf_read;
    }
    else
    {
        num = buf_write + (sizeof(buffer) - buf_read);
    }

    event->SetEvent();
    lock->Unlock();

    if (num<0 || num > sizeof(buffer))
    {
        AfxAbort();
```

```
    }

    return num;
}

////////////////////////////////////////////////////////////////////////////
// Serial message handlers
```

B.3.11 UnComp Class

B.3.11.1 UnComp.h

```
// UnComp.h : header file
//

////////////////////////////////////////////////////////////////////////////
// UnComp object

class UnComp
{
// Attributes
public:
private:
    HLOCAL m_BitMem;
    HLOCAL m_BitGroups;
    int m_totalgroups;
    int m_usedgroups;
    int m_linespergroup;
    int m_width;
    int m_totallines;
    int m_usedlines;

// Operations
public:
    UnComp();
    ~UnComp();
    UnComp(int width);
    UnComp(int width,int groups);
    void Create(int width,int groups);
    void Destroy();
    void Paint(CDC* pDC);
    BOOL UnComp::CanCopyAll();
    void CopyAll(CDC *pDC);
    int GetTotalLines();
    int GetWidth();
    BYTE *GrabLine(int line, HLOCAL *handle, BOOL expand);
    int GetValue(int x, int y);
};

class UnCompLine
{
// Attributes
public:
    BYTE *m_pointer;
private:
    HLOCAL m_handle;

// Operations
```

```
public:
    UnCompLine(UnComp *base, int line, BOOL expand);
    ~UnCompLine();
};

////////////////////////////////////////////////////////////////////////////
```

B.3.11.2 UnComp.cpp

```
// UnComp.cpp : implementation file
//

#include "stdafx.h"
#include "larch.h"
#include "UnComp.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// constant definitions

#define DEFAULTWIDTH 1024
#define DEFAULTGROUPS 4

#define INCREMENTGROUPS 4

////////////////////////////////////////////////////////////////////////////
// UnComp

// Create default compressed data segments
UnComp::UnComp()
{
    Create(DEFAULTWIDTH,DEFAULTGROUPS);
}

// Create compressed data segments with specifications
UnComp::UnComp(int width)
{
    Create(width,DEFAULTGROUPS);
}

// Create compressed data segments with more specifications
UnComp::UnComp(int width,int numgroups)
{
    Create(width, numgroups);
}

// Constructor
void UnComp::Create(int width,int numgroups)
{
    int i,j;
    BITMAPINFO *bmi;
    HLOCAL *groups;

    // Figure out the number of segments we need
    m_width=width;
```

```
    m_linespergroup=65536/m_width;

    // Allocate the global memory
    m_BitMem = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE, sizeof(BITMAPINFOHEADER)
+ (sizeof(RGBQUAD) * 256));

    // Create a gray scale palette
    bmi = (PBITMAPINFO) LocalLock(m_BitMem);
    for (i = 0; i < 256; i++)
    {
        bmi->bmiColors[i].rgbRed=LOBYTE(i);
        bmi->bmiColors[i].rgbGreen=LOBYTE(i);
        bmi->bmiColors[i].rgbBlue=LOBYTE(i);
        bmi->bmiColors[i].rgbReserved=0;
    }

    // Create the bitmap info
    bmi->bmiHeader.biSize=sizeof(BITMAPINFOHEADER);
    bmi->bmiHeader.biWidth=width; // sizes
    bmi->bmiHeader.biHeight=m_linespergroup; // sizes
    bmi->bmiHeader.biPlanes=1;
    bmi->bmiHeader.biBitCount=8;
    bmi->bmiHeader.biCompression=BI_RGB;
    bmi->bmiHeader.biSizeImage=0;
//  bmi->bmiHeader.biXPelsPerMeter=GetDeviceCaps(hdc,HORZRES)/
GetDeviceCaps(hdc,HORZSIZE);
//  bmi->bmiHeader.biYPelsPerMeter=GetDeviceCaps(hdc,VERTRES)/
GetDeviceCaps(hdc,VERTSIZE);
    bmi->bmiHeader.biClrUsed=256;
    bmi->bmiHeader.biClrImportant=256;

    LocalUnlock(m_BitMem);

    // (SDI documents will reuse this document)

    // Allocate all group memory segments
    m_totalgroups = numgroups;
    m_usedgroups = 0;
    m_usedlines = -1;

    m_BitGroups = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
sizeof(groups)*m_totalgroups);
    groups = (HLOCAL*) LocalLock(m_BitGroups);

    for (j=0;j<m_totalgroups;j++)
    {
       groups[j] = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
m_width*m_linespergroup);

       // For debugging
/*
       data = (PBYTE) LocalLock(groups[j]);

       for (i=0;i<m_width*m_linespergroup;i++)
          data[i]=(i + ((i >> 8)) + j*64) & 255;

       LocalUnlock(groups[j]);
*/
```

```
    }

    // Release pointers
    LocalUnlock(m_BitGroups);

}

// Destructor
UnComp::~UnComp()
{
    Destroy();
}

// Unallocate all the memory
void UnComp::Destroy()
{
    int j;
    HLOCAL *groups;

    // Release groups
    groups = (HLOCAL*) LocalLock(m_BitGroups);
    for (j=0;j<m_totalgroups;j++)
    {
        LocalFree(groups[j]);
    }
    LocalUnlock(m_BitGroups);
    // Release pointers
    LocalFree(m_BitGroups);

    LocalFree(m_BitMem);
}

// Paint the user screen
void UnComp::Paint(CDC* pDC)
{
    BITMAPINFO *bmi;
    BYTE *data;
    HLOCAL *groups;
    int j,gs,gf,i,k,gl,last;
    CRect reg,draw;

    last = m_usedlines - 1;

    if (last<0) return;

    pDC->GetClipBox(&reg);

    bmi = (PBITMAPINFO) LocalLock(m_BitMem);
    groups = (HLOCAL*) LocalLock(m_BitGroups);

    gs = reg.TopLeft().y / m_linespergroup;
    gf = min(reg.BottomRight().y,last) / m_linespergroup;
    gl = last / m_linespergroup;

//  TRACE2("Refresh: %d %d\n",gs,gf);

    // Figure out segments to repaint
    for (j=gs;j<=gf;j++)
    {
```

```
        k = 0;
        i = m_linespergroup;

        if (j==gl)
        {
            i = (last % m_linespergroup) + 1;
            k = m_linespergroup - i;
        }

        data = (PBYTE) LocalLock(groups[j]);
        SetDIBitsToDevice(pDC->m_hDC,
            reg.TopLeft().x,j*m_linespergroup,
            reg.Width(),i,
            reg.TopLeft().x,0,
            0,i,
            data+k*m_width,bmi,DIB_RGB_COLORS);
        LocalUnlock(groups[j]);
    }

    LocalUnlock(m_BitGroups);
    LocalUnlock(m_BitMem);

    // Debug test
//  CString text("This is a test!");
//  pDC->TextOut(0,0,text);

}

// Can we do a copy all?
BOOL UnComp::CanCopyAll()
{
    if (m_usedlines > 0)
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

// Copy all segments to the Windows clipboard
void UnComp::CopyAll(CDC *pDC)
{
    TRACE0("UnComp::CopyAll()\n");

    CWaitCursor wait;

    BITMAPINFO *bmi,*gbmi;
    HGLOBAL bitmem;
    BYTE *data,*sdata;
    HANDLE test;
    HLOCAL *groups;
    int last;

    last = m_usedlines - 1;

    bitmem = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE | GMEM_DDESHARE ,
LocalSize(m_BitMem) + (m_width * m_usedlines) );
```

```
    TRACE1("UnComp::CopyAll(bitmem(size)=%d)\n",GlobalSize(bitmem));

    bmi = (PBITMAPINFO) LocalLock(m_BitMem);
    gbmi = (PBITMAPINFO) GlobalLock(bitmem);

    memcpy(gbmi,bmi,LocalSize(m_BitMem));

    LocalUnlock(m_BitMem);

    gbmi->bmiHeader.biHeight = last;
    gbmi->bmiHeader.biSizeImage = 0; //m_width * m_usedlines;
    gbmi->bmiHeader.biXPelsPerMeter=pDC->GetDeviceCaps(HORZRES) / pDC-
>GetDeviceCaps(HORZSIZE);
    gbmi->bmiHeader.biYPelsPerMeter=pDC->GetDeviceCaps(VERTRES) / pDC-
>GetDeviceCaps(VERTSIZE);

    data = (BYTE*) gbmi + LocalSize(m_BitMem);

    int j,i,k,gl;

    gl = last / m_linespergroup;

    groups = (HLOCAL*) LocalLock(m_BitGroups);

    for (j=gl;j>=0;j--)
    {
        sdata = (PBYTE) LocalLock(groups[j]);

        if (j!=gl)
        {
            memcpy(data,sdata,LocalSize(groups[j]));
            data += LocalSize(groups[j]);
        }
        else
        {
            i = (last % m_linespergroup) + 1;
            k = m_linespergroup - i;

            memcpy(data,sdata+(k*m_width),i*m_width);
            data += (i*m_width);

        }

        LocalUnlock(groups[j]);

    }

    LocalUnlock(m_BitGroups);

    GlobalUnlock(bitmem);

    test = SetClipboardData(CF_DIB, bitmem);

    return;
}

// Get total lines of image, to maintain private information
int UnComp::GetTotalLines()
```

```
{
    return m_totallines;
}

// Get image width
int UnComp::GetWidth()
{
    return m_width;
}

// Get a line from an image
BYTE *UnComp::GrabLine(int line, HLOCAL *handle, BOOL expand)
{
    int i,j;
    HLOCAL *groups;
    BYTE *pointer;

    *handle = NULL;

    if (line >= m_totalgroups*m_linespergroup)
    {
        if (expand==TRUE)
        {
            i = line/m_linespergroup + INCREMENTGROUPS;

            m_BitGroups = LocalReAlloc(m_BitGroups,  sizeof(groups)*i,
LMEM_ZEROINIT | LMEM_MOVEABLE);

            groups = (HLOCAL*) LocalLock(m_BitGroups);

            for (j=m_totalgroups;j<i;j++)
            {
                groups[j] = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
m_width*m_linespergroup);
            }

            LocalUnlock(m_BitGroups);

            m_totalgroups=i;
        }
        else
        {
            return NULL;
        }
    }

    if (line > m_usedlines)
    {
        if (expand == FALSE)
        {
            return NULL;
        }
    }

    groups = (HLOCAL*) LocalLock(m_BitGroups);

    *handle = groups[line/m_linespergroup];
    pointer = (PBYTE) LocalLock(*handle)+(m_linespergroup - 1 - (line %
m_linespergroup))*m_width;
```

```
    LocalUnlock(m_BitGroups);

    m_totallines = max(line,m_totallines);
    m_usedlines = max(line,m_usedlines);

    if (pointer==NULL)
    {
        AfxAbort();
    }

    return pointer;
}

// Get a value to update gray level on status bar
int UnComp::GetValue(int x, int y)
{
    UnCompLine *line;
    int value;

    line = new UnCompLine(this, y, FALSE);

    if (line->m_pointer==NULL)
    {
        value=-1;
    }
    else
    {
        value=line->m_pointer[x];
    }

    delete line;

    return value;
}

////////////////////////////////////////////////////////////////////////////

// Cache line to minimize memory access
UnCompLine::UnCompLine(UnComp* base, int line, BOOL expand)
{
    m_pointer=base->GrabLine(line, &m_handle, expand);
}

// Destructor, release windows resource
UnCompLine::~UnCompLine()
{
    if (m_pointer!=NULL)
    {
        LocalUnlock(m_handle);
    }
    m_pointer=NULL;
}

////////////////////////////////////////////////////////////////////////////
```

B.3.12 Comp Class

B.3.12.1 Comp.h

```
// Comp.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// Comp object

class Comp
{
// Attributes
public:
protected:
    HLOCAL m_BitGroups;
    int m_totalgroups;
    int m_bytespergroup;
    int m_totalbytes;
    int m_processedbytes;
    UnComp *m_uncompdata;
    int m_uncompline;
    int m_uncomppixel;
    int m_cachegroup;
    HLOCAL m_cachehandle;
    BYTE *m_cachepointer;

// Operations
public:
//  Comp(UnComp *uncomp);
//  ~Comp();
//  Comp(UnComp *uncomp, int size);
    void Serialize(CArchive& ar);
    void Create(UnComp *uncomp, int groups);
    void Destroy();
    void AddData(BYTE *data, int length);
    void InvalidateCache();
    int GetByte();
    CSize GetSize();

    virtual void InitData()=0;
    virtual void EndData()=0;
    virtual void ProcessData()=0;
};

/////////////////////////////////////////////////////////////////////////////
```

B.3.12.2 Comp.cpp

```
// Comp.cpp : implementation file
//
// Based on UnComp.cpp

#include "stdafx.h"
#include "larch.h"
#include "UnComp.h"
#include "Comp.h"

#ifdef _DEBUG
```

```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// constant definitions

#define GROUPSIZE 32768
#define DEFAULTGROUPS 5

#define INCREMENTGROUPS 2

////////////////////////////////////////////////////////////////////////////
// UnComp

// Creator, segments to hold compressed data, more efficient than realloc
void Comp::Create(UnComp *uncomp, int numgroups)
{
    int j;
    HLOCAL *groups;

    m_uncompdata = uncomp;
    m_bytespergroup = GROUPSIZE;
    m_totalbytes = 0;
    m_processedbytes = 0;
    m_uncompline=0;
    m_uncomppixel=0;
    m_cachegroup = -1;
    m_cachepointer = NULL;
    m_cachehandle = NULL;
    // (SDI documents will reuse this document)

    m_totalgroups = numgroups;

    m_BitGroups = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
sizeof(groups)*m_totalgroups);
    groups = (HLOCAL*) LocalLock(m_BitGroups);

    for (j=0;j<m_totalgroups;j++)
    {
        groups[j] = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE, m_bytespergroup);
    }

    LocalUnlock(m_BitGroups);

    InitData();

}

// Destory object
void Comp::Destroy()
{
    InvalidateCache();

    int j;
    HLOCAL *groups;

    groups = (HLOCAL*) LocalLock(m_BitGroups);
    for (j=0;j<m_totalgroups;j++)
```

```
    {
        LocalFree(groups[j]);
    }
    LocalUnlock(m_BitGroups);
    LocalFree(m_BitGroups);

    return;
}

// Load and Save data
void Comp::Serialize(CArchive& ar)
{
    InvalidateCache();

    int length;

    if (ar.IsStoring())
    {
        ar << (DWORD) m_totalbytes;
        ar << (DWORD) m_uncompdata->GetWidth();

        HLOCAL *groups;

        groups = (HLOCAL*) LocalLock(m_BitGroups);

        int j;
        BYTE *pointer;

        for (j=0;j<=m_totalbytes/m_bytespergroup;j++)
        {
            pointer = (BYTE*) LocalLock(groups[j]);
            length = m_bytespergroup;
            if (m_totalbytes/m_bytespergroup == j)
            {
                length = m_totalbytes % m_bytespergroup;
            }

            ar.Write(pointer,length);

            LocalUnlock(groups[j]);
        }

        LocalUnlock(m_BitGroups);
    }
    else
    {
        DWORD total,width;
        HLOCAL handle;
        BYTE *buffer;

        handle = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE, m_bytespergroup/2);
        buffer = (BYTE*) LocalLock(handle);

        ar >> (DWORD) total;
        ar >> (DWORD) width;

        do
        {
            length=ar.Read(buffer,min((DWORD)m_bytespergroup/2,total));
```

```
            if (length>0)
            {
                AddData(buffer,length);
            }
            total -= length;
        }
        while (length>0 || total>0 );

        ProcessData();
        EndData();

        LocalUnlock(handle);
        LocalFree(handle);
    }
}

// Add data and more segments if necessary
void Comp::AddData(BYTE *data, int length)
{
    int i,j;
    HLOCAL *groups;
    HLOCAL handle;
    BYTE *pointer;

    InvalidateCache();

    if (length > m_bytespergroup)
    {
        AfxAbort();
    }

    if (length+m_totalbytes >= m_totalgroups*m_bytespergroup)
    {
        i = m_totalgroups + INCREMENTGROUPS;

        m_BitGroups = LocalReAlloc(m_BitGroups,  sizeof(groups)*i, LMEM_ZEROINIT |
LMEM_MOVEABLE);

        groups = (HLOCAL*) LocalLock(m_BitGroups);

        for (j=m_totalgroups;j<i;j++)
        {
            groups[j] = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
m_bytespergroup);
        }

        LocalUnlock(m_BitGroups);

        m_totalgroups=i;
    }

    groups = (HLOCAL*) LocalLock(m_BitGroups);

    handle = groups[m_totalbytes/m_bytespergroup];

    pointer = (PBYTE) LocalLock(handle)+(m_totalbytes%m_bytespergroup);

    if ( (m_totalbytes/m_bytespergroup) == ((m_totalbytes+length)/
m_bytespergroup) )
```

```
    {
        memcpy(pointer,data,length);
    }
    else
    {
        i=m_bytespergroup-(m_totalbytes % m_bytespergroup);
        memcpy(pointer,data,i);
        LocalUnlock(handle);
        handle = groups[(m_totalbytes+length)/m_bytespergroup];
        pointer = (PBYTE) LocalLock(handle);
        memcpy(pointer,data+i,length-i);
    }

    m_totalbytes += length;

    LocalUnlock(handle);
    LocalUnlock(m_BitGroups);

}

// Get a byte from the segment (serially)
int Comp::GetByte()
{
    HLOCAL *groups;
    HLOCAL handle;
    BYTE *pointer;
    BYTE value;
    int index;

    if (m_processedbytes>m_totalbytes)
    {
        return -1;
//      AfxAbort();
    }

    index = m_processedbytes / m_bytespergroup;

    if (index == m_cachegroup)
    {

        value = m_cachepointer[m_processedbytes%m_bytespergroup];

    }
    else
    {
        InvalidateCache();

        groups = (HLOCAL*) LocalLock(m_BitGroups);

        handle = groups[ index ];

        pointer = (PBYTE) LocalLock(handle);

        value = pointer[ m_processedbytes % m_bytespergroup ];

        m_cachepointer = pointer;
        m_cachegroup = index;
        m_cachehandle = handle;
```

```
//      LocalUnlock(handle);

        LocalUnlock(m_BitGroups);

    }

    m_processedbytes += 1;

//  TRACE1("%x ",value);

    return value;
}

// Invalidate cache for higher speed
void Comp::InvalidateCache()
{
    if (m_cachehandle != NULL)
    {
        LocalUnlock(m_cachehandle);
    }

    m_cachegroup = -1;
    m_cachepointer = NULL;
    m_cachehandle = NULL;
}

// Get size of allocated data
CSize Comp::GetSize()
{
    CSize size(m_uncompdata->GetWidth(),max(m_uncompline - 1,0));

    return size;
}

////////////////////////////////////////////////////////////////////////////
```

B.3.13 Comp1:Comp Class

B.3.13.1 Comp1.h

```
// Comp1.h : header file
//

////////////////////////////////////////////////////////////////////////////
// Comp1 object

class Comp1 : public Comp
{
// Attributes
public:
private:
    BOOL m_synced;
    int m_synccount;

// Operations
public:
    Comp1(UnComp *uncomp);
    ~Comp1();
```

```
    Compl(UnComp *uncomp, int size);

    void InitData();
    void EndData();
    void ProcessData();
    BOOL SyncData();
};

//////////////////////////////////////////////////////////////////////////////
```

B.3.13.2 Compl.cpp

```
// Compl.cpp : implementation file
//
// Decompressor for Modifier RLE data compression

#include "stdafx.h"
#include "larch.h"
#include "UnComp.h"
#include "Comp.h"
#include "Compl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// constant definitions

#define GROUPSIZE 32768
#define DEFAULTGROUPS 5

#define INCREMENTGROUPS 2

//////////////////////////////////////////////////////////////////////////////
// Compl

// Special constructor
Compl::Compl(UnComp *uncomp)
{
    Create(uncomp, DEFAULTGROUPS);
}

// Special constructor
Compl::Compl(UnComp *uncomp, int numgroups)
{
    Create(uncomp, numgroups);
}

// Special desructor
Compl::~Compl()
{
    Destroy();
}

// Initialize data before capture
void Compl::InitData()
{
```

```
    m_synced=FALSE;
    m_synccount=0;
}

// Finish up after data is received
void Compl::EndData()
{
    m_synced=FALSE;
    m_synccount=0;
}

// Synchronize data to beginning of a line
BOOL Compl::SyncData()
{
    int value;
    int temp=m_processedbytes;

    do
    {
        do
        {
            value=GetByte();
            if (value<0)
            {
                break;
            }
        }
        while (value!=0);
        if (value<0)
        {
            break;
        }

        value=GetByte();
        if (value<0)
        {
            break;
        }
    }
    while (value!=0);

    if (value<0)
    {
        m_processedbytes=temp;
        return FALSE;
    }

    do
    {
        value=GetByte();
        if (value<0)
        {
            break;
        }
    }
    while (value==0);

    if (value<0)
    {
```

```
        m_processedbytes=temp;
        return FALSE;
    }

    m_processedbytes--;
    m_synced=TRUE;

    return TRUE;
}

// Decompress the data
void Compl::ProcessData()
{
    int i,j;
    BYTE value_a,value_b,value_c;
    BOOL needsync=FALSE;
    UnCompLine *line;
    int mx=m_uncompdata->GetWidth()-1;

//  TRACE0("Sync: ");

    if (m_synced==FALSE)
    {
            if (SyncData()==FALSE)
            {
                return;
            }
    }

    line=new UnCompLine(m_uncompdata, m_uncompline, TRUE);

//  TRACE0("\nData: ");

    while (m_processedbytes+50<m_totalbytes)
    {
        value_a = GetByte();
        value_b = GetByte();

        if (value_a!=0)
        {
            if (value_a!=255)
            {
                for(i=0;i<value_a;i++)
                {
                    line->m_pointer[min(m_uncomppixel+i,mx)]=value_b;
                }
                m_uncomppixel+=i;
            }
            else
            {
                for(i=0;i<254;i++)
                {
                    line->m_pointer[min(m_uncomppixel+i,mx)]=value_b;
                }
                m_uncomppixel+=i;
                do
                {
                    value_a = value_c = GetByte();
                    if (value_a==255) value_c = 254;
```

```
                for(i=0;i<value_c;i++)
                {
                    line->m_pointer[min(m_uncomppixel+i,mx)]=value_b;
                }
                m_uncomppixel+=i;
            }
            while (value_a==255);
        }
    }
    else
    {
        if (m_uncomppixel!=0)
        {
            needsync=TRUE;
            break;
        }
        m_synccount=0;
    }

    if (m_uncomppixel==m_uncompdata->GetWidth())
    {
        delete line;
        m_uncomppixel=0;
        m_uncompline++;
        m_synccount++;
        line=new UnCompLine(m_uncompdata, m_uncompline, TRUE);
//      TRACE0("\n");
    }
    else if (m_uncomppixel>m_uncompdata->GetWidth())
    {
        needsync=TRUE;
        break;
    }
}

// TRACE0("\n");
   delete line;

   if (needsync==TRUE)
   {
       m_synced=FALSE;
       m_uncomppixel=0;

       for (j=0;j<16;j++)
       {
           line=new UnCompLine(m_uncompdata, m_uncompline, TRUE);
           for (i=0;i<m_uncompdata->GetWidth();i++)
           {
               line->m_pointer[i]= ( (i + m_uncompline) & 1 ) ? 0 : 255;
           }
           delete line;
           m_uncompline++;
       }
   }

}

//////////////////////////////////////////////////////////////////////////////
```

B.3.14 CGetParams Class

B.3.14.1 GetParams.h

```
// GetParams.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CGetParams dialog

#define GETPARAMS_MAX 8

class CGetParams : public CDialog
{
private:
    CStatic *m_text[GETPARAMS_MAX];
    CEdit *m_edit[GETPARAMS_MAX];
    CSpinButtonCtrl *m_spin[GETPARAMS_MAX];
    BYTE *m_prcblock,*m_rules;
    BYTE m_total;
// Construction
public:
    void SetParams(BYTE *params);
    void GetParams(BYTE *params);
    CGetParams(CWnd* pParent = NULL, BYTE* prcblock = NULL);   // standard
constructor
    void UseDefaultParams();
    BYTE GetCompressor();
    int GetLineLength();
    int GetVersionMajor();
    int GetVersionMinor();
    CString GetTitle();
    BOOL IsValid();

// Dialog Data
    //{{AFX_DATA(CGetParams)
    enum { IDD = IDD_CAPTURE_PARAMS };
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CGetParams)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CGetParams)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BYTE m_params[GETPARAMS_MAX];
};
```

B.3.14.2 GetParams.cpp

```
// GetParams.cpp : implementation file
//

#include "stdafx.h"
#include "larch.h"
//#include "GetParams.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGetParams dialog


CGetParams::CGetParams(CWnd* pParent, BYTE *prcblock)
    : CDialog(CGetParams::IDD, pParent)
{
    //{{AFX_DATA_INIT(CGetParams)
    //}}AFX_DATA_INIT
    m_prcblock = prcblock;
}


void CGetParams::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CGetParams)
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CGetParams, CDialog)
    //{{AFX_MSG_MAP(CGetParams)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGetParams message handlers


void CGetParams::GetParams(BYTE *params)
{
    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        params[i]=m_params[i];
    }
}

void CGetParams::SetParams(BYTE *params)
{
    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
```

```
    {
        m_params[i]=params[i];
    }
}

BOOL CGetParams::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here
    static int i_text[GETPARAMS_MAX] = {IDC_TEXT1, IDC_TEXT2, IDC_TEXT3,
IDC_TEXT4,
        IDC_TEXT5, IDC_TEXT6, IDC_TEXT7, IDC_TEXT8};
    static int i_edit[GETPARAMS_MAX] = {IDC_EDIT1, IDC_EDIT2, IDC_EDIT3,
IDC_EDIT4,
        IDC_EDIT5, IDC_EDIT6, IDC_EDIT7, IDC_EDIT8};
    static int i_spin[GETPARAMS_MAX] = {IDC_SPIN1, IDC_SPIN2, IDC_SPIN3,
IDC_SPIN4,
        IDC_SPIN5, IDC_SPIN6, IDC_SPIN7, IDC_SPIN8};

    int i;

    BYTE *point = m_prcblock+9;

    SetDlgItemText(IDC_TITLE,(LPCTSTR)point);
    point += (strlen((LPCTSTR)point)+1);

    m_total = *point;
    point++;

    // point is at default values

    point += m_total;

    for (i=0;i<GETPARAMS_MAX;i++)
    {
        m_text[i] = (CStatic*)GetDlgItem(i_text[i]);
        m_edit[i] = (CEdit*)GetDlgItem(i_edit[i]);
        m_spin[i] = (CSpinButtonCtrl*)GetDlgItem(i_spin[i]);

        m_edit[i]->LimitText(3);
        m_edit[i]->SetWindowText("");

        m_spin[i]->SetRange(0,255);
        m_spin[i]->SetPos(m_params[i]);
    }

    for (i=0;i<m_total;i++)
    {
        m_text[i]->EnableWindow(TRUE);
        m_edit[i]->EnableWindow(TRUE);
        m_spin[i]->EnableWindow(TRUE);

        m_text[i]->SetWindowTe..  .LPCTSTR)point);
        point += (strlen((LPCTSTR)point)+1);
    }

    for (i=m_total;i<GETPARAMS_MAX;i++)
    {
```

```
        m_spin[i]->SetPos(0);
    }

    int up,low;
    m_rules = point;

    while (*point!=255)
    {
        i = point[0];
        m_spin[i]->GetRange(low,up);

        switch (point[1])
        {
            case 0: // greater than static
                low = point[2]+1;
                break;
            case 1: // lesser than static
                up = point[2]-1;
                break;
            case 2: // greater than equal static
                low = point[2];
                break;
            case 3: // lesser than equal static
                up = point[2];
                break;
        }
        m_spin[i]->SetRange(low,up);

        point+=3;
    }

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CGetParams::OnOK()
{
    // TODO: Add extra validation here
    int i;
    BYTE *point = m_rules;
    BYTE value1,value2;
    BOOL error;
    CString title1,title2,problem,message;

    while (*point!=255)
    {
        error = FALSE;
        value1 = m_spin[point[0]]->GetPos();

        switch (point[1])
        {
            case 4: // greater than index
                value2 = m_spin[point[2]]->GetPos();
                if (value1 <= value2)
                {
                    problem = "greater than";
                    error = TRUE;
                }
                break;
```

```
            case 5: // lesser than index
                value2 = m_spin[point[2]]->GetPos();
                if (value1 >= value2)
                {
                    problem = "lesser than";
                    error = TRUE;
                }
                break;
            case 6: // greater than equal index
                value2 = m_spin[point[2]]->GetPos();
                if (value1 < value2)
                {
                    problem = "greater than or equal to";
                    error = TRUE;
                }
                break;
            case 7: // lesser than equal index
                value2 = m_spin[point[2]]->GetPos();
                if (value1 > value2)
                {
                    problem = "lesser than or equal to";
                    error = TRUE;
                }
                break;
        }
        if (error == TRUE)
        {
            m_text[point[0]]->GetWindowText(title1);
            m_text[point[2]]->GetWindowText(title2);
            message = "'" + title1 + "' must be " + problem + " '" + title2 + "'.";
            MessageBox(message,"Parameter Error",MB_ICONEXCLAMATION);
            break;
        }

        point+=3;
    }

    if (error==FALSE)
    {
        for (i=0;i<GETPARAMS_MAX;i++)
        {
            m_params[i] = m_spin[i]->GetPos();
        }

        CDialog::OnOK();
    }

}

void CGetParams::UseDefaultParams()
{
    BYTE *point = m_prcblock+9;

    point += (strlen((LPCTSTR)point)+1);

    int total = *point;
    point++;

    int i;
```

```
    for (i=0;i<total;i++)
    {
        m_params[i] = point[i];
    }
}

BYTE CGetParams::GetCompressor()
{
    if (IsValid()==FALSE)
    {
        return 0;
    }

    return m_prcblock[6];
}

int CGetParams::GetVersionMajor()
{
    return m_prcblock[3] * 256 + m_prcblock[2];
}

int CGetParams::GetVersionMinor()
{
    return m_prcblock[5] * 256 + m_prcblock[4];
}

int CGetParams::GetLineLength()
{
    if (IsValid()==FALSE)
    {
        return -1;
    }
    else
    {
        return m_prcblock[8] * 256 + m_prcblock[7];
    }
}

CString CGetParams::GetTitle()
{
    if (IsValid()==FALSE)
    {
        CString title("");
        return title;
    }
    else
    {
        CString title(m_prcblock+9);
        return title;
    }
}


BOOL CGetParams::IsValid()
{
    if (m_prcblock[0] == 0 && m_prcblock[1] == 0)
    {
        return FALSE;
    }
```

```
    return TRUE;
}
```

B.3.15 Video Processor Headers

The video processors are located in "FPG" files. These files are a combination of the video processing parameters and the FPGA bitmap. The video processing parameter files (HDR) are shown here. Each of these binary files are concatenated with the FPGA bitstream to create the FPG file.

B.3.15.1 Deltatracker.hdr

(Binary Data)

```
$000  55 57 5F 4C 41 52 43 48 ; Application Magic Number: UW_LARCH
$008  46 50 47 41             ; Processor Magic Number: FPGA
$00C  01 00 00 00             ; Version: 1.0.0.0
$010  00 62                   ; Data Remaining: 98 bytes
$012  01 00 01 00             ; Processor ID
$016  01                      ; Compressor Type: 1
$017  00 04                   ; Line Width: 1024
$019  44 65 6C 74 61 54 72 61
$021  63 6B 65 72 20 56 69 64
$029  65 6F 20 50 72 6F 63 65
$031  73 73 6F 72 00          ; Title: Threshold Video Processor
$036  03                      ; Number of Parameters: 3
$037  00 0A 02                ; Defaults: 0, 10, 2
$03A  42 61 63 6B 67 72 6F 75
$042  6E 64 00                ; P1: Background
$045  4D 61 78 69 6D 75 6D 20
$04D  64 65 76 69 61 74 69 6F
$055  6E 00                   ; P2: Maximum deviation
$057  42 61 6E 64 77 69 64 74
$05F  68 00                   ; P3: Bandwidth
$061  00 02 00                ; P1 >= 0
$064  00 03 FF                ; P1 <= 255
$067  01 00 00                ; P2 >  0
$06A  01 01 80                ; P2 <  128
$06D  02 00 00                ; P3 >  0
$070  02 03 0F                ; P3 <  15
$073  FF
```

B.3.15.2 Maxmin.hdr

(Binary Data)

```
$000  55 57 5F 4C 41 52 43 48 ; Application Magic Number: UW_LARCH
$008  46 50 47 41             ; Processor Magic Number: FPGA
$00C  01 00 00 00             ; Version: 1.0.0.0
$010  00 6B                   ; Data Remaining: 107
$012  02 00 01 00             ; Processor ID
$016  01                      ; Compressor Type: 1
$017  00 04                   ; Line Width: 1024
$019  4D 61 78 69 6D 75 6D 2F
$021  4D 69 6E 69 6D 75 6D 20
$029  56 69 64 65 6F 20 50 72 ; Title: Maximum/Minimum
$031  6F 63 65 73 73 6F 72 00 ;        Video Processor
$039  03                      ; Number of Parameters: 3
$03A  00 CD 7D                ; Defaults: 0, 205, 125
$03D  42 61 63 6B 67 72 6F 75
$045  6E 64 00                ; P1: Background
$048  4D 61 78 69 6D 75 6D 20
$050  4C 65 76 65 6C 00       ; P2: Maximum Level
$056  4D 69 6E 69 6D 75 6D 20
$05E  4C 65 76 65 6C 00       ; P3: Minimum Level
$064  00 02 00                ; P1 >= 0
$067  00 03 FF                ; P1 <= 255
$06A  01 02 00                ; P2 >= 0
$06D  01 03 FF                ; P2 <= 255
$070  02 02 00                ; P3 >= 0
$073  02 03 FF                ; P3 <= 255
$076  01 04 02                ; P2 >  P3
$079  02 05 01                ; P3 <  P2
$07C  FF
```

B.3.15.3 Threshold.hdr

(Binary Data)

```
$000  55 57 5F 4C 41 52 43 48 ; Application Magic Number: UW_LARCH
$008  46 50 47 41             ; Processor Magic Number: FPGA
$00C  01 00 00 00             ; Version: 1.0.0.0
$010  00 3C                   ; Data Remaining: 60 bytes
$012  03 00 01 00             ; Processor ID
$016  01                      ; Compressor Type: 1
$017  00 04                   ; Line Width: 1024
$019  54 68 72 65 73 68 6F 6C
$021  64 20 56 69 64 65 6F 20
$029  50 72 6F 63 65 73 73 6F
$031  72 00                   ; Title: Threshold Video Processor
$033  01                      ; Number of Parameters: 1
$034  05                      ; Defaults: 5
$035  4D 61 78 69 6D 75 6D 20
$03D  64 65 76 69 61 74 69 6F
$045  6E 00                   ; P1: Maximum deviation
$047  00 02 00                ; P1 >  0
$04A  00 03 FF                ; P1 <= 255
$04D  FF
```

B.3.15.4 Range.hdr

(Binary Data)

```
$000  55 57 5F 4C 41 52 43 48 ; Application Magic Number: UW_LARCH
$008  46 50 47 41             ; Processor Magic Number: FPGA
$00C  01 00 00 00             ; Version: 1.0.0.0
$010  00 5D                   ; Data Remaining: 93 bytes
$012  04 00 01 00             ; Processor ID
$016  01                      ; Compressor Type: 1
$017  00 04                   ; Line Width: 1024
$019  52 61 6E 67 65 20 56 69
$021  64 65 6F 20 50 72 6F 63
$029  65 73 73 6F 72 00       ; Title: Range Video Processor
$02F  03                      ; Number of Parameters: 3
$030  00 CD 7D                ; Defaults: 0, 205, 125
$033  42 61 63 6B 67 72 6F 75
$03B  6E 64 00                ; P1: Background
$03E  55 70 70 65 72 20 4C 65
$046  76 65 6C 00             ; P2: Upper Level
$04A  4C 6F 77 65 72 20 4C 65
$052  76 65 6C 00             ; P3: Lower Level
$056  00 02 00                ; P1 >= 0
$059  00 03 FF                ; P1 <= 255
$05C  01 02 00                ; P2 >= 0
$05F  01 03 FF                ; P2 <= 255
$062  02 02 00                ; P3 >= 0
$065  02 03 FF                ; P3 <= 255
$068  01 04 02                ; P2 >  P3
$06B  02 05 01                ; P3 <  P2
$06E  FF
```

Appendix C

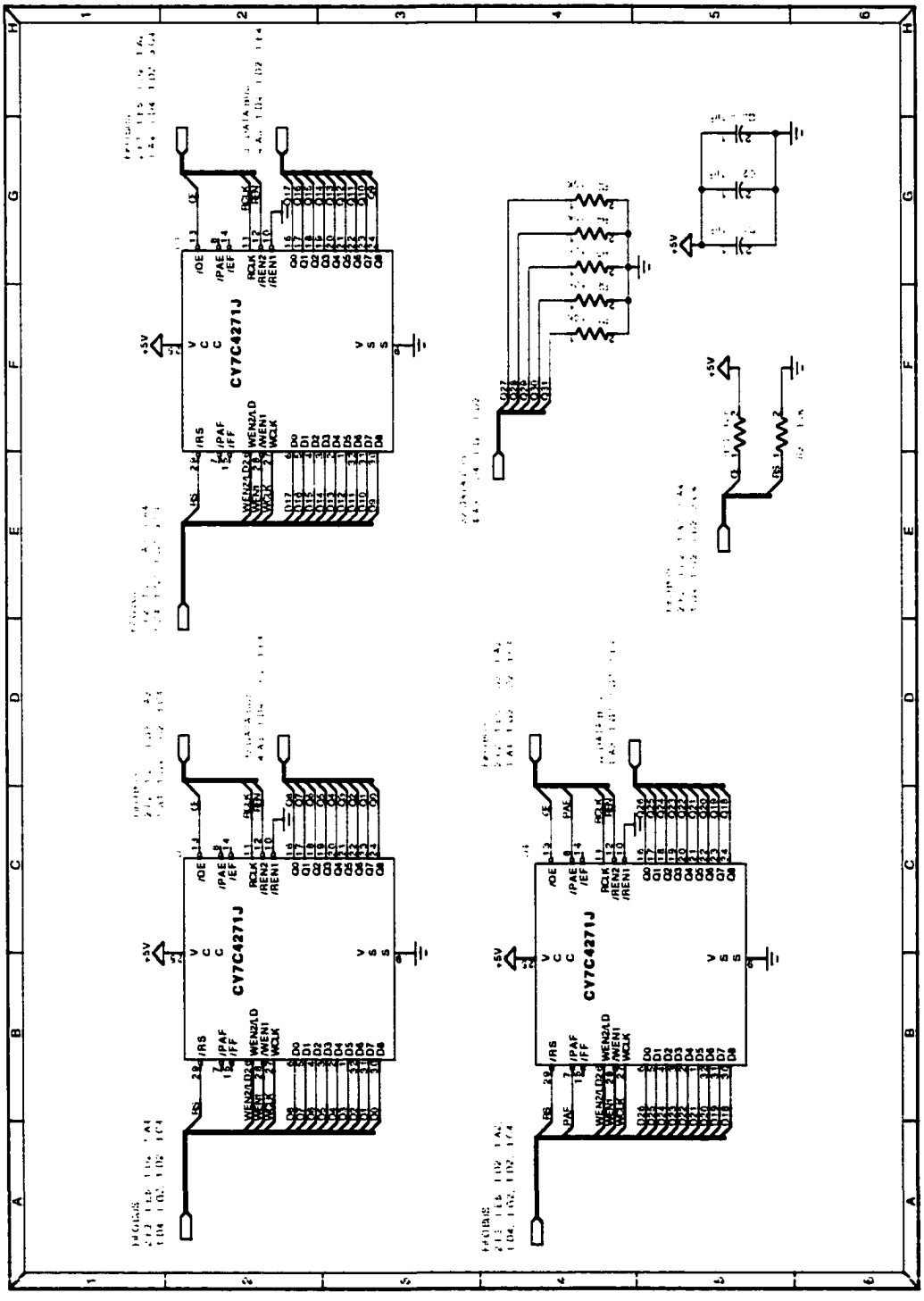
Second Generation System Hardware

This appendix contains the hardware design for the second generation system. As in the first generation system, the is in a DesignWorks v3.1.5 schematic capture database (Appendix C.1 "Schematics" on page 300) where it is used to verify with the layout (Appendix C.2 "PCB Layouts" on page 315) generated manually with the Douglas Professional Layout v4.6 tool. The layout is used to manufacture the PCB (Appendix C.3 "Fabricated Boards" on page 318) by Douglas' PCB Manufacturing Facilities. The assembled board (with the help of Dalsa Inc.) is shown in Appendix C.4 "Assembled Boards" on page 321. The final system can be found in Appendix C.4.3.2 "Solder Side" on page 323.

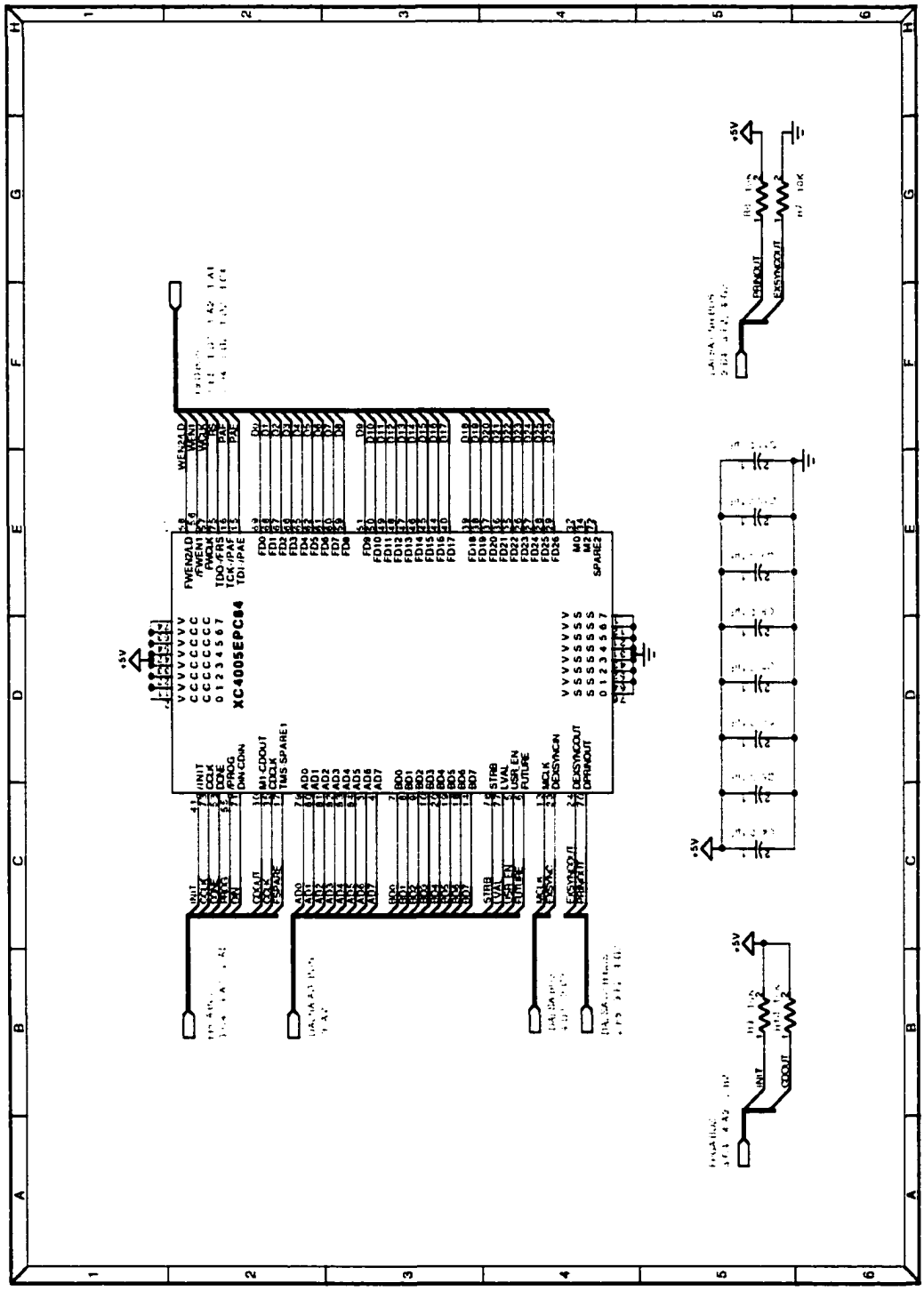
C.1 Schematics

C.1.1 FPGA

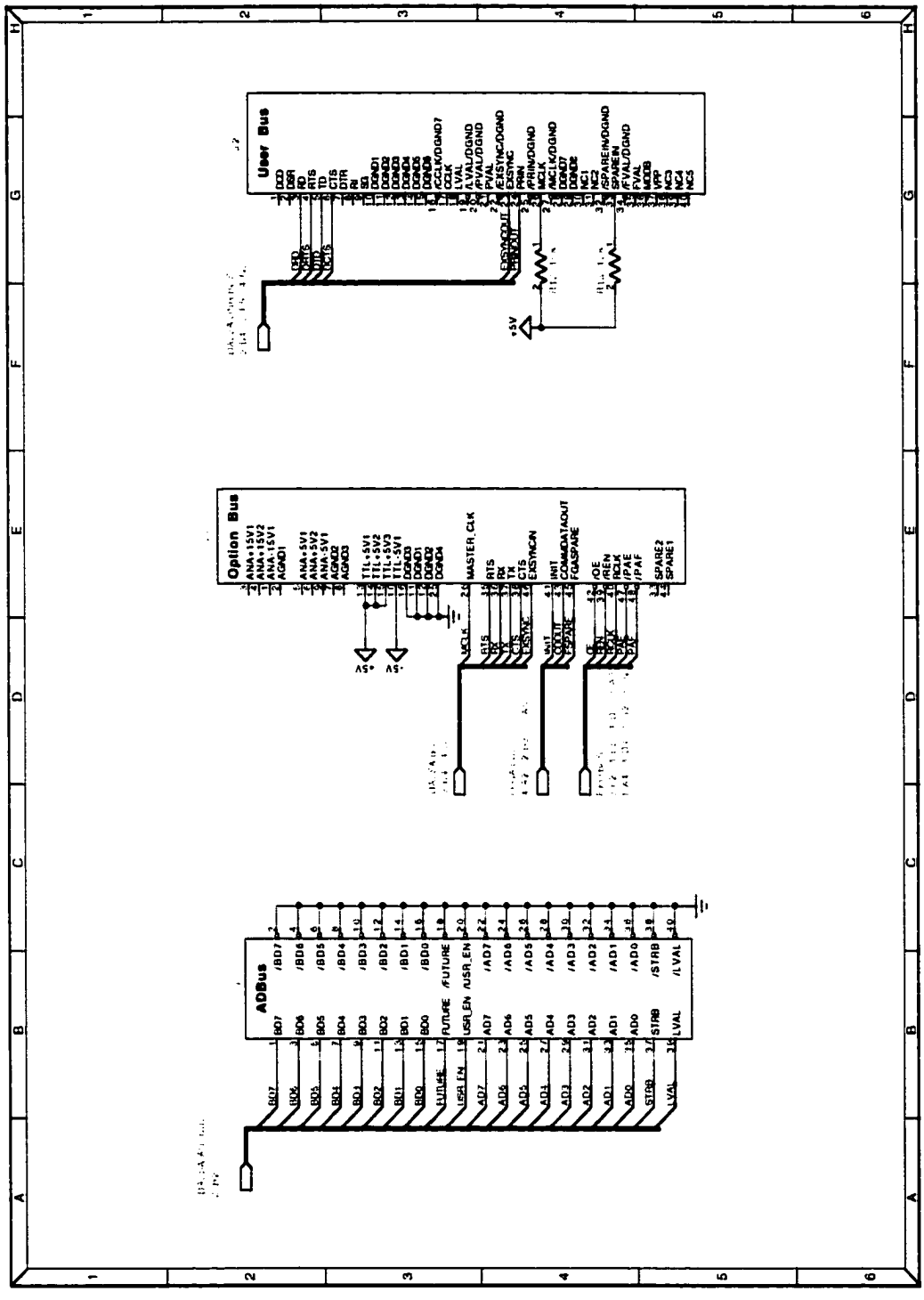
C.1.1.1 Page 1



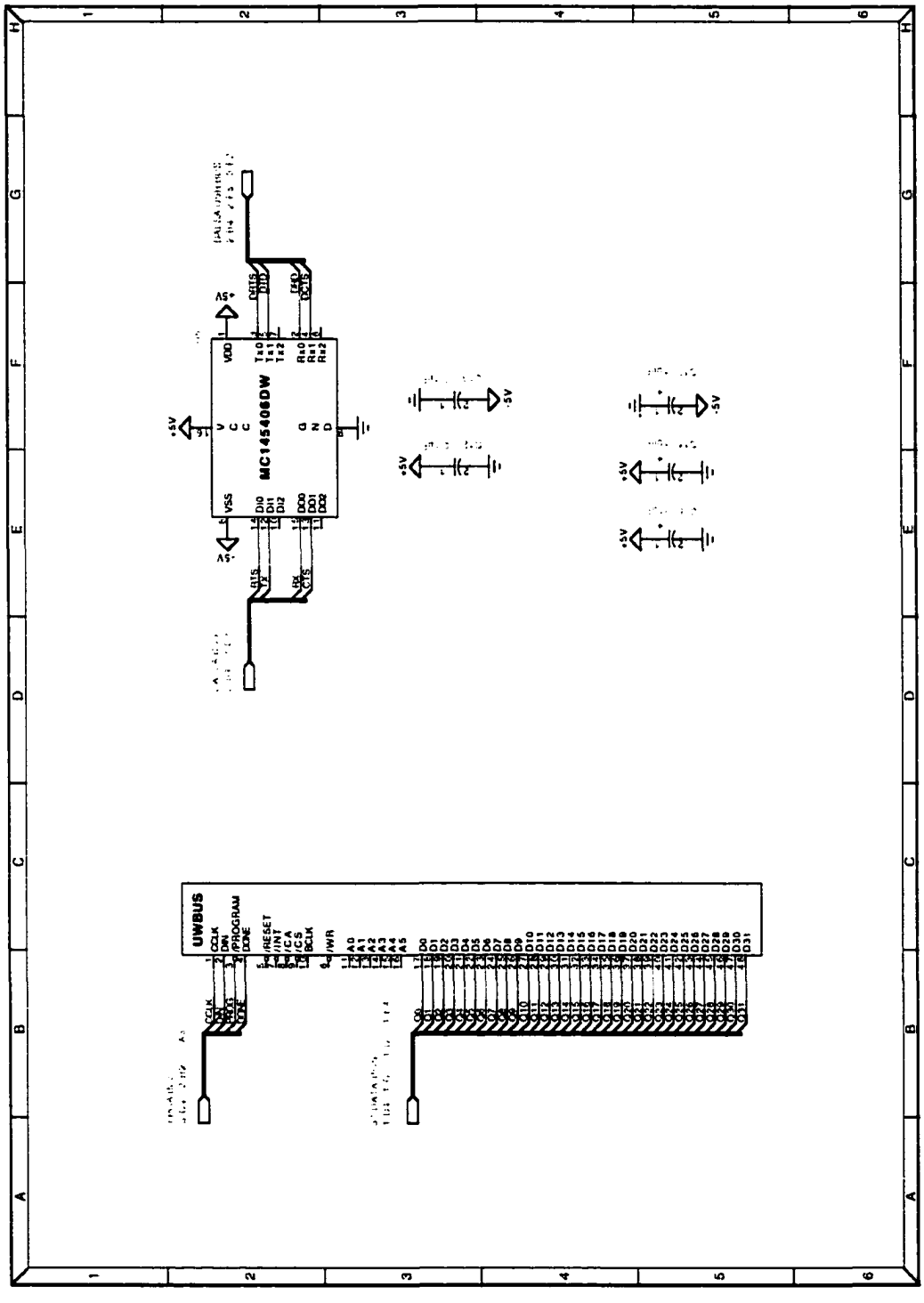
C.1.1.2 Page 2



C.1.1.3 Page 3

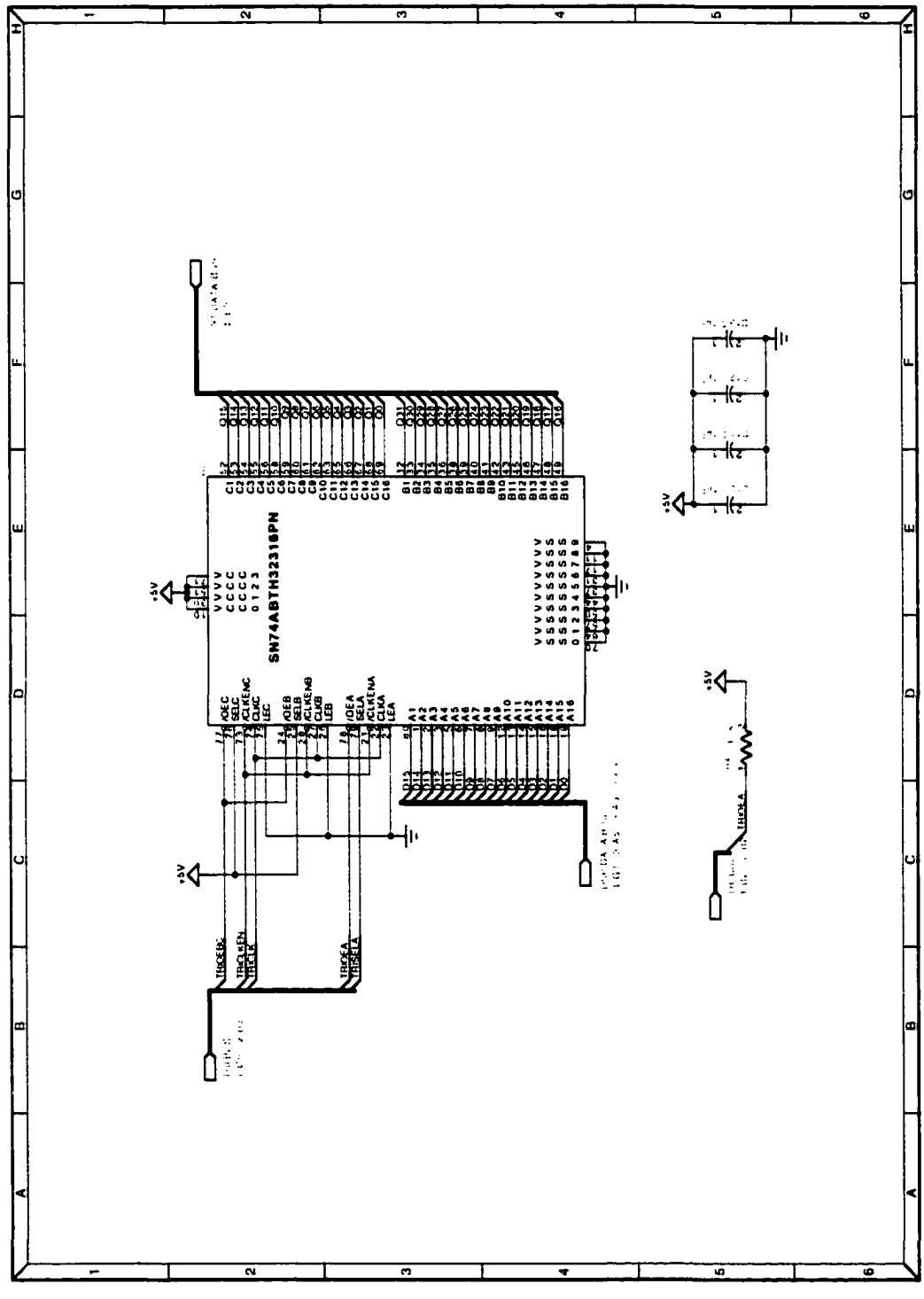


C.1.1.4 Page 4

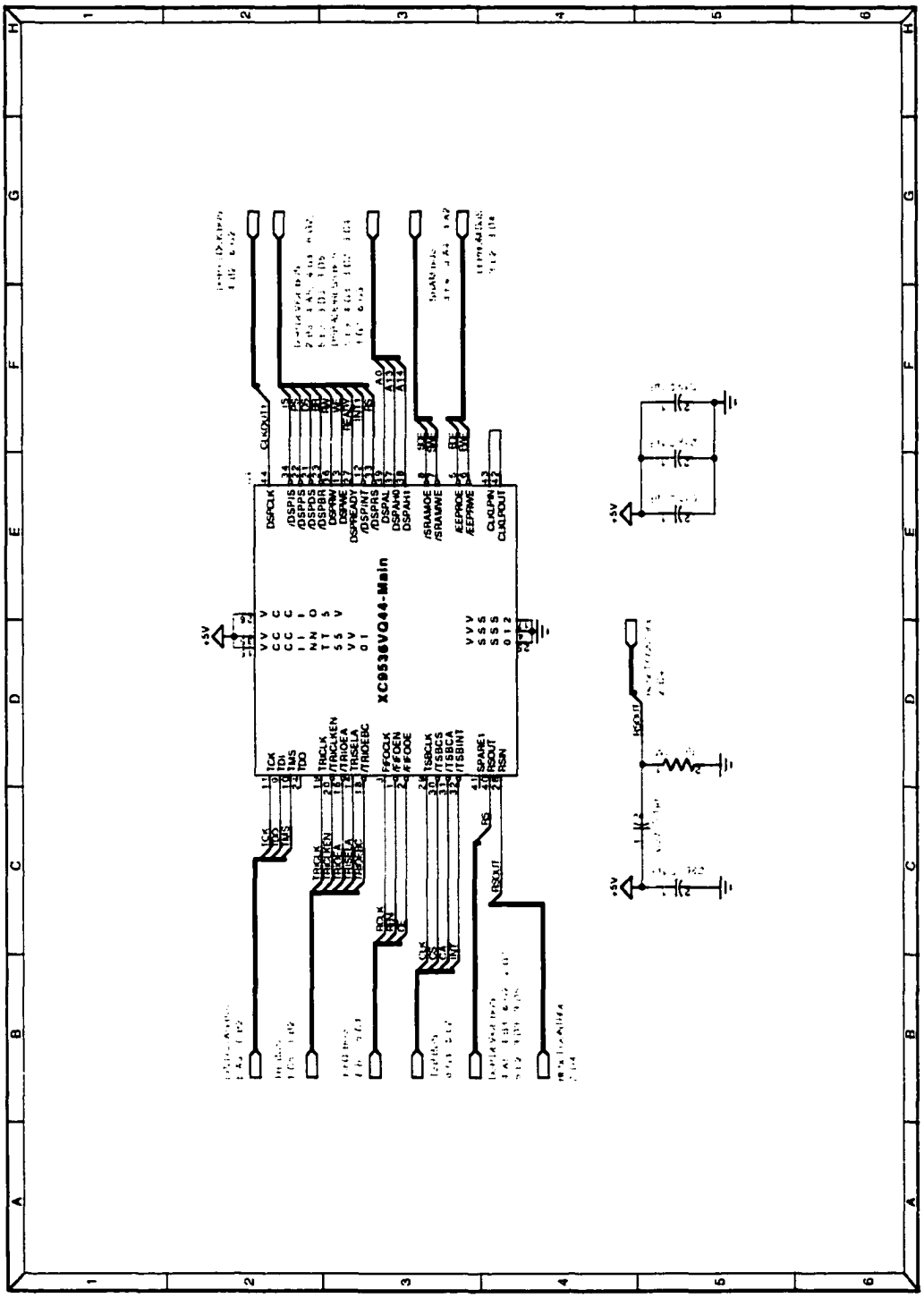


C.1.2 DSP

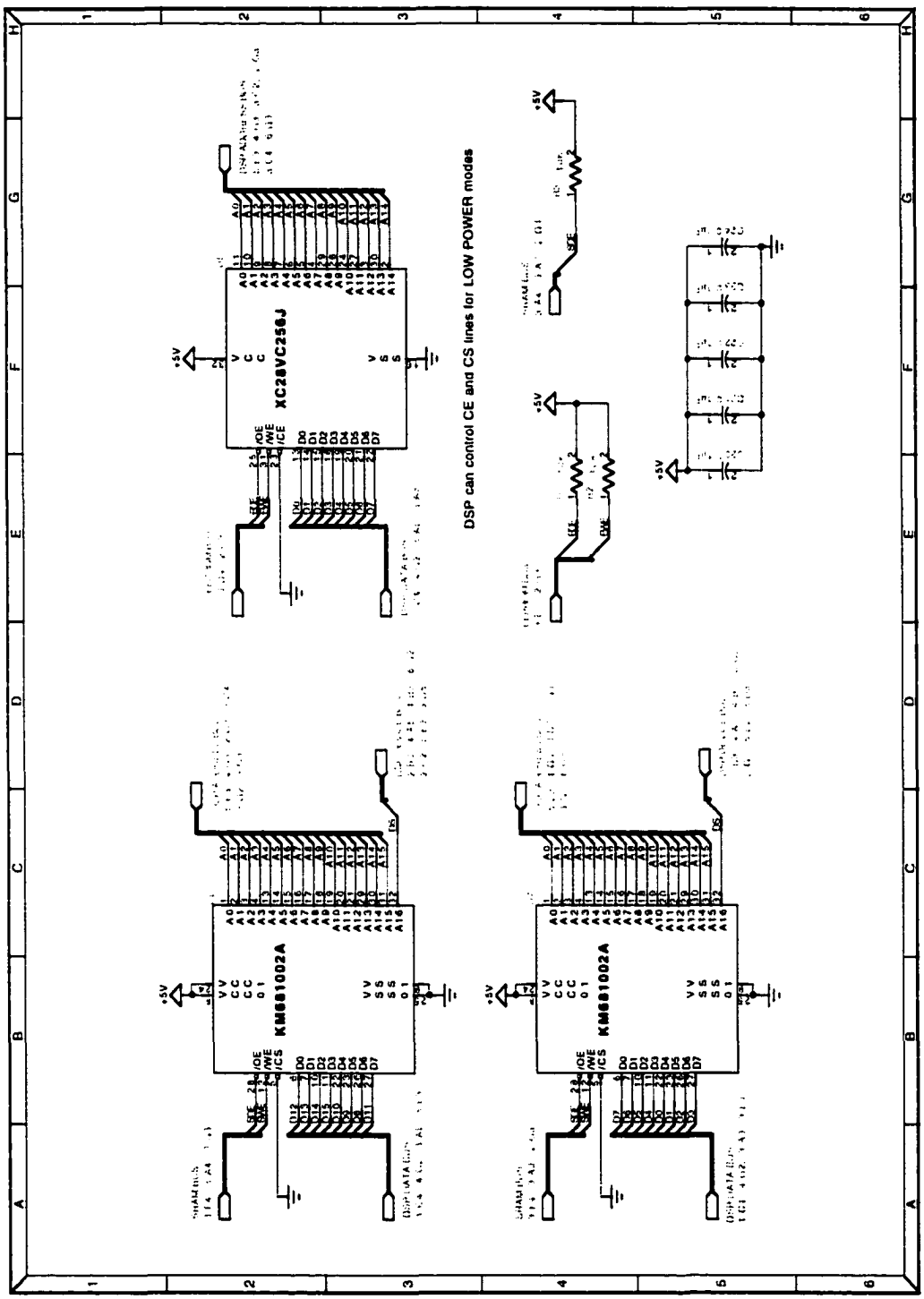
C.1.2.1 Page 1



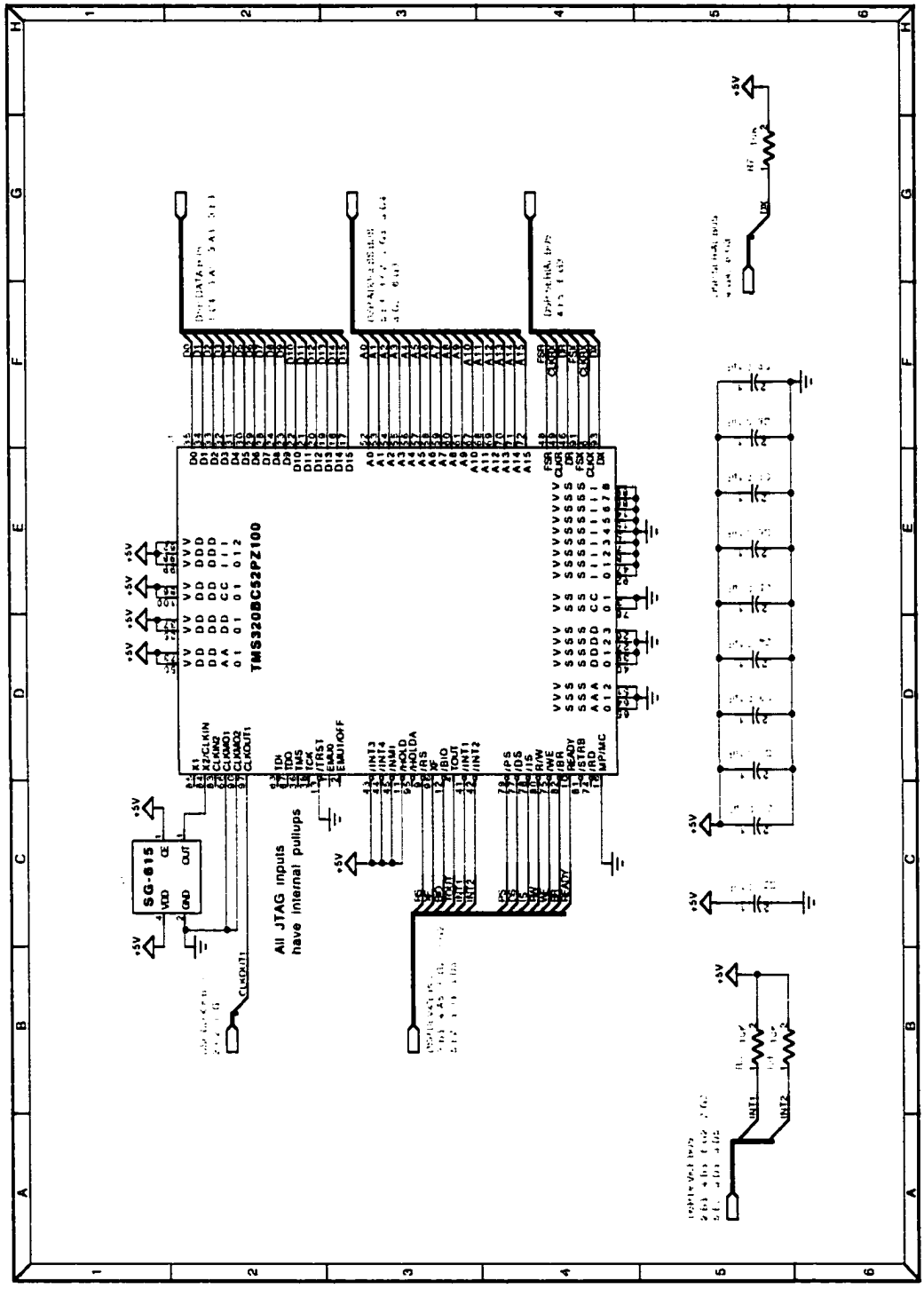
C.1.2.2 Page 2



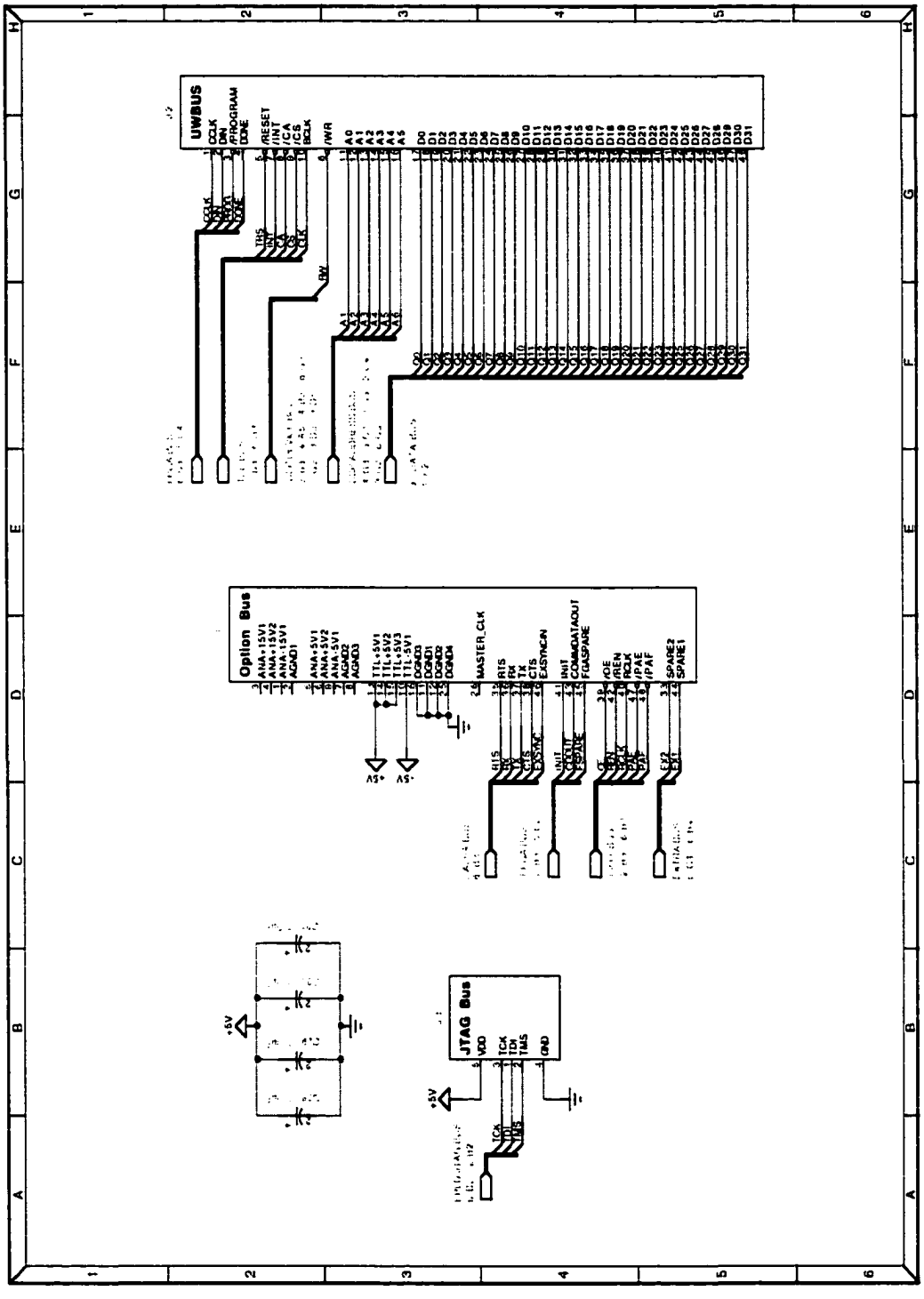
C.1.2.3 Page 3



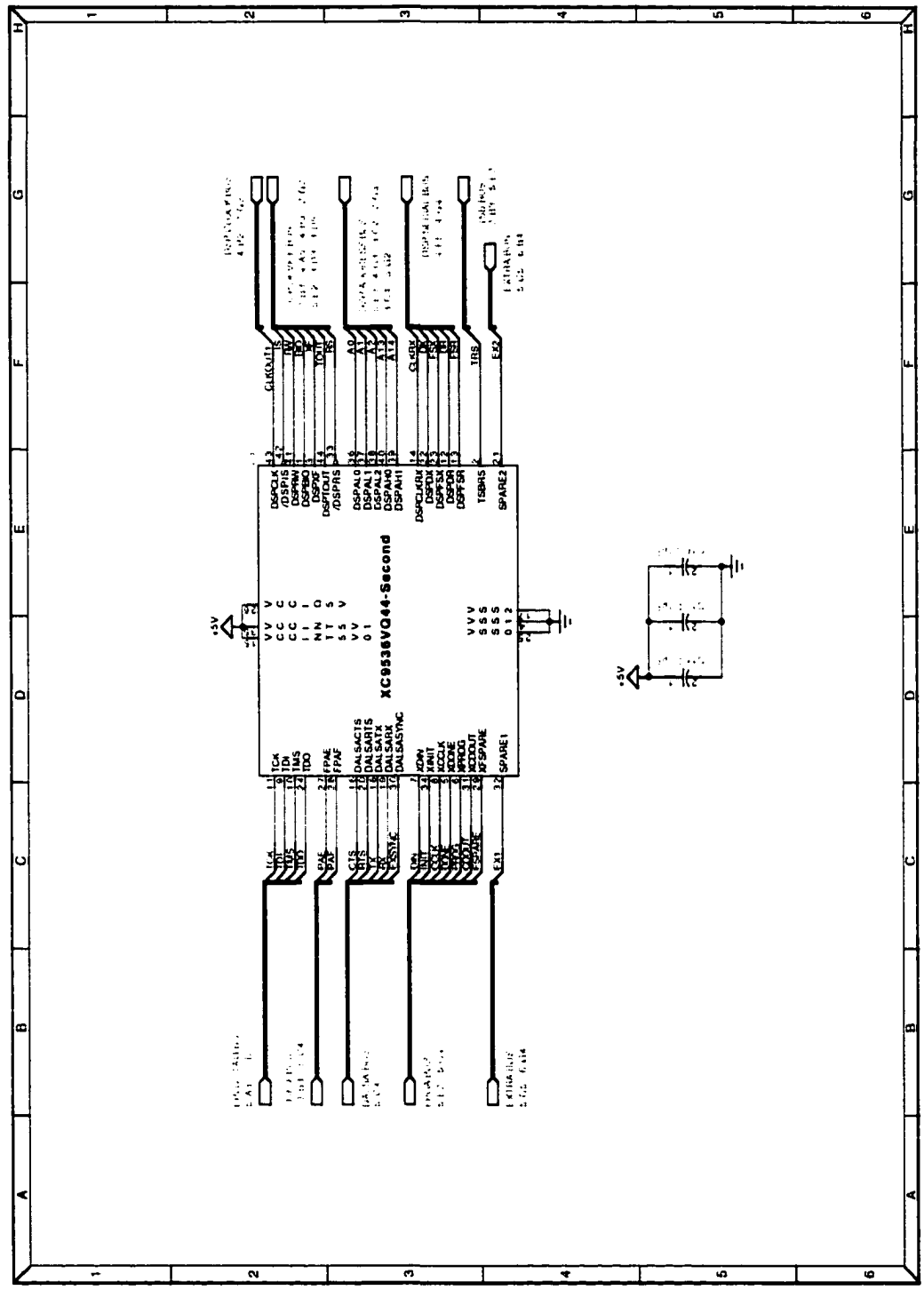
C.1.2.4 Page 4



C.1.2.5 Page 5

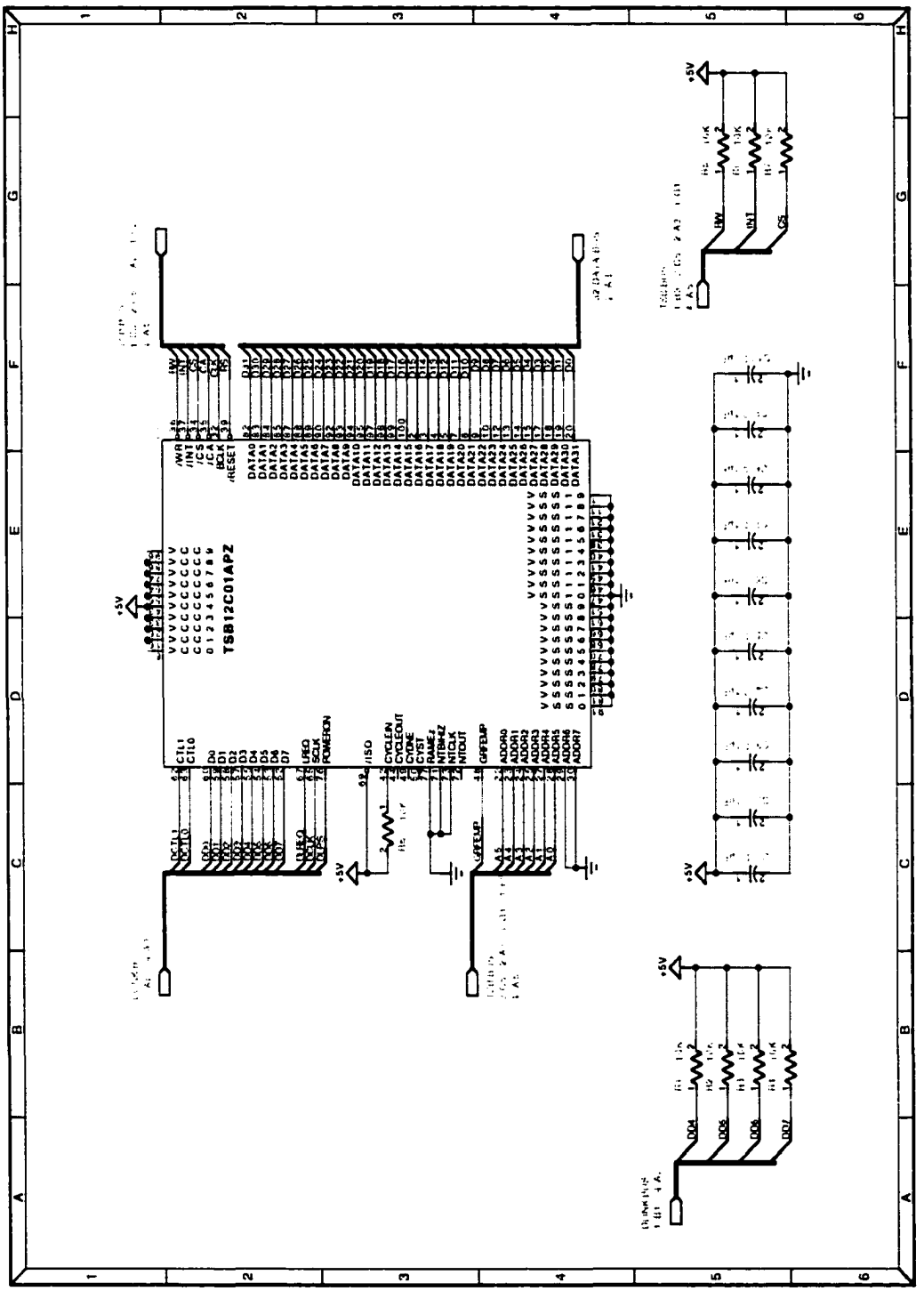


C.1.2.6 Page 6

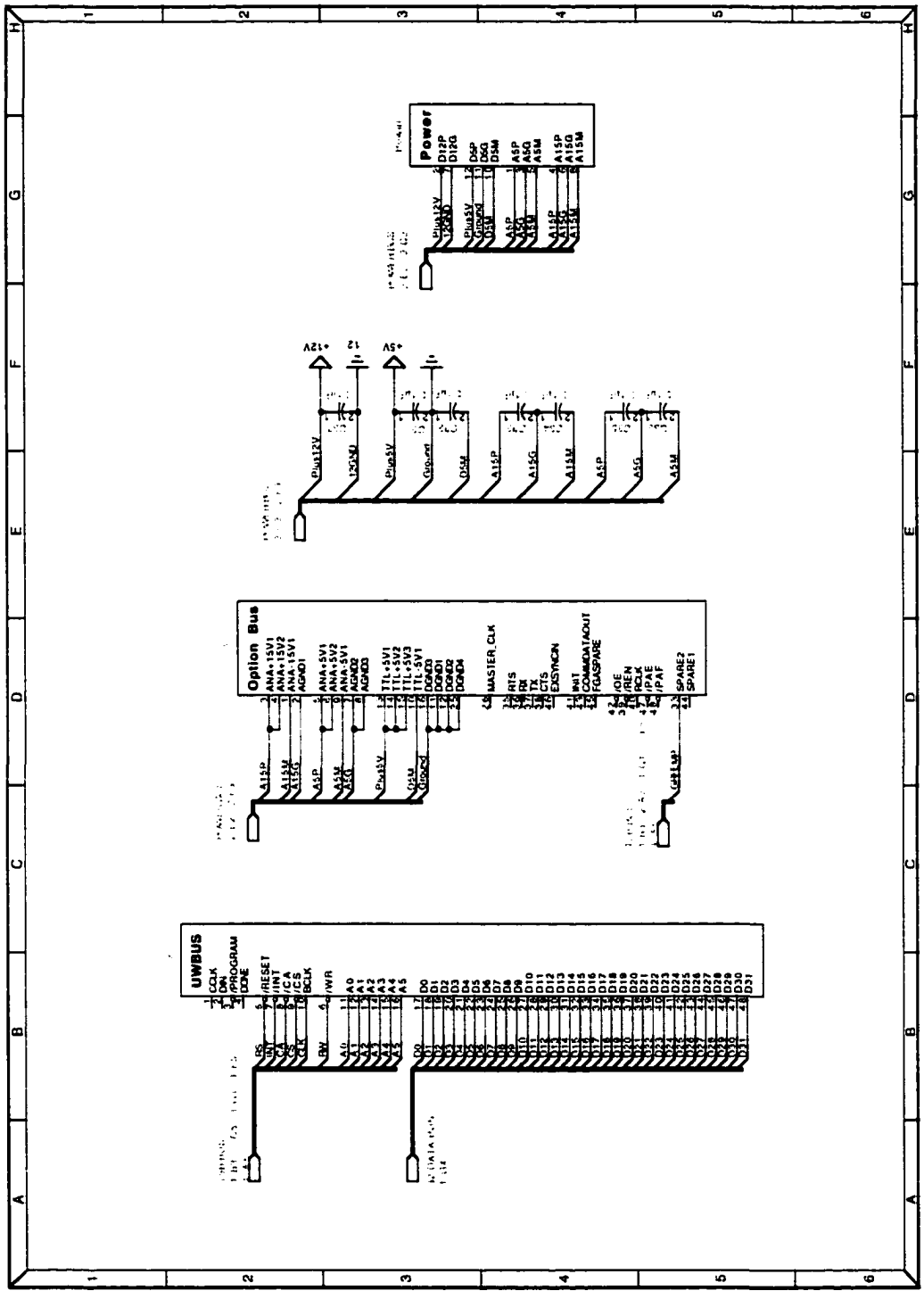


C.1.3 1394

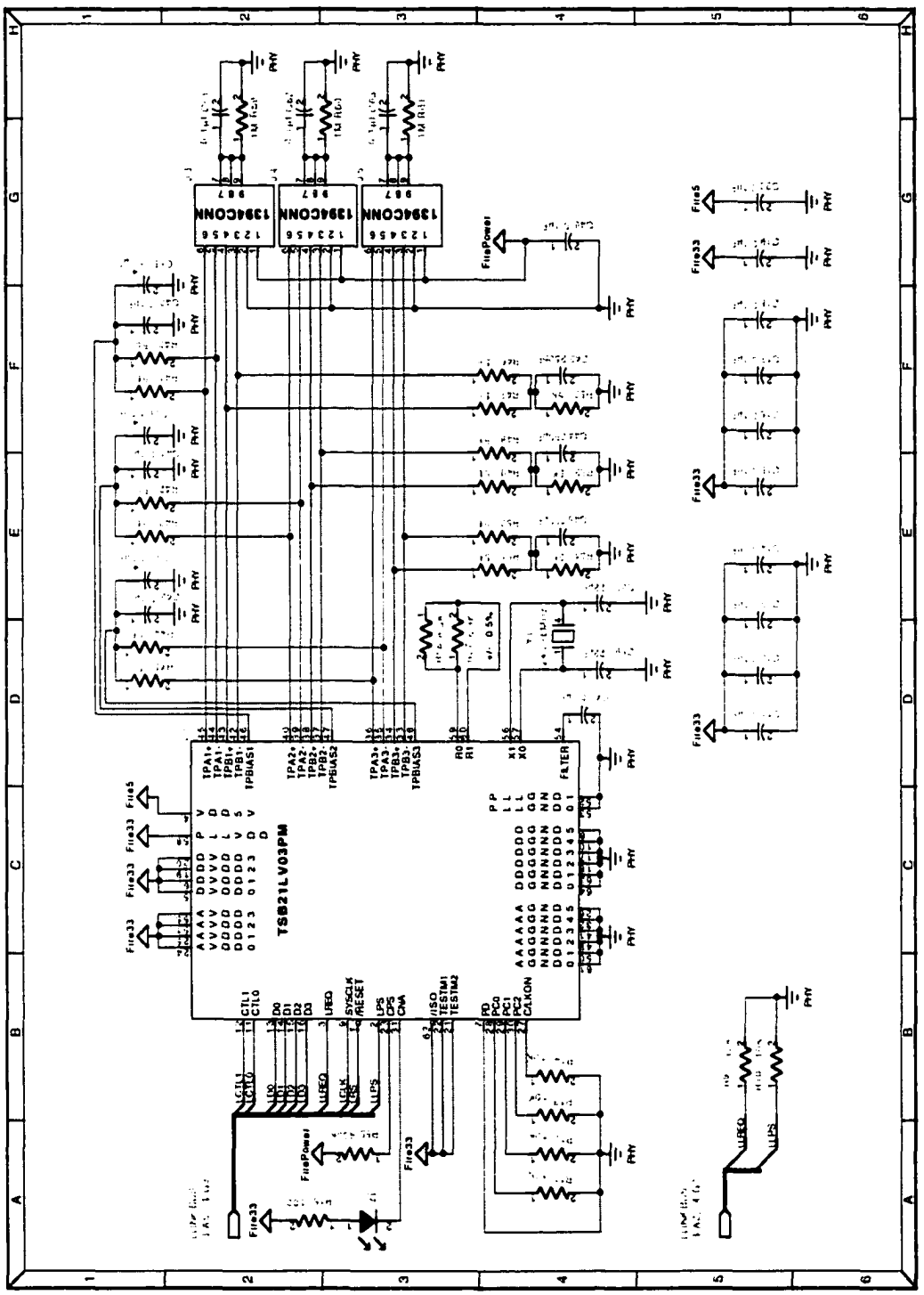
C.1.3.1 Page 1



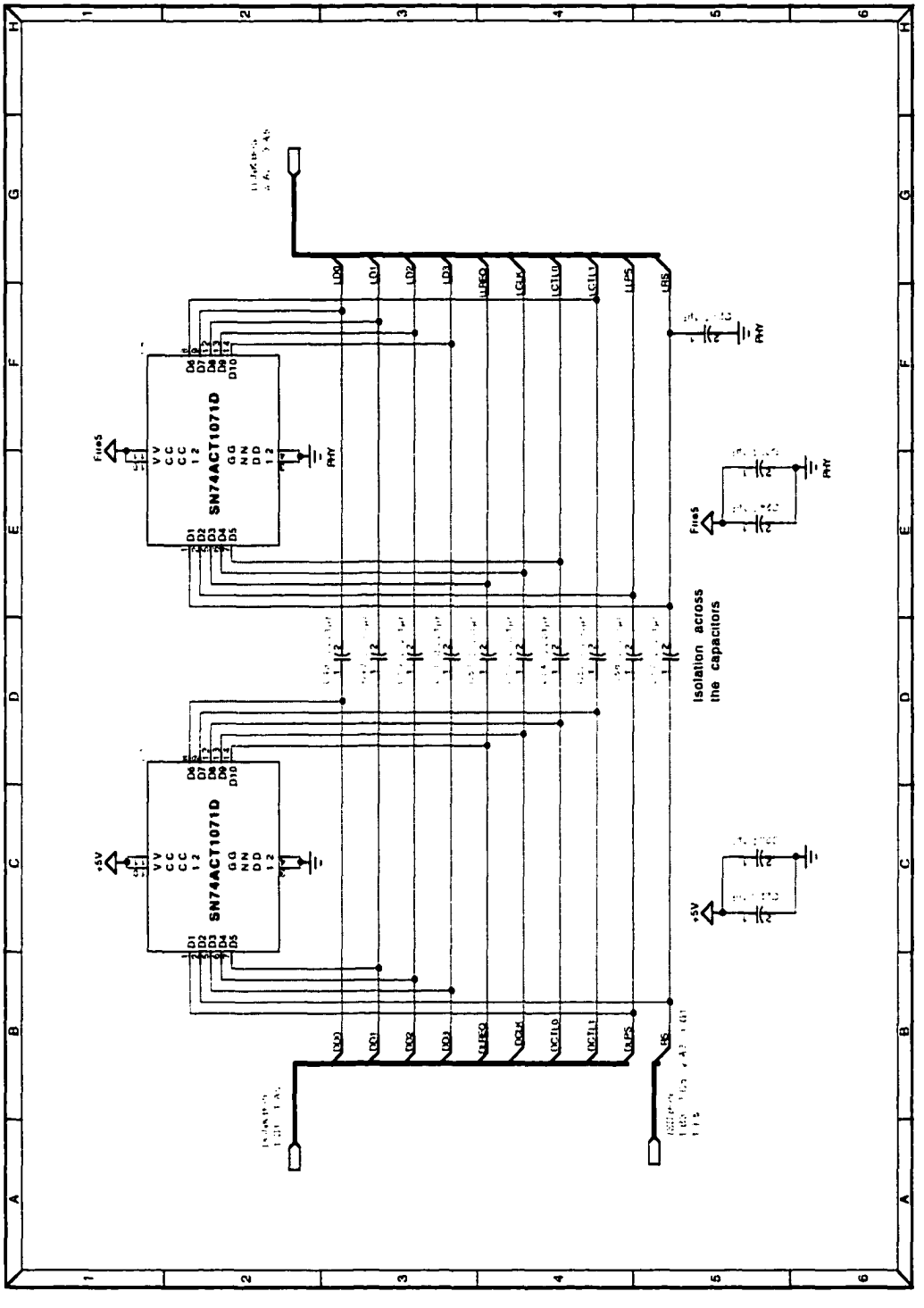
C.1.3.2 Page 2



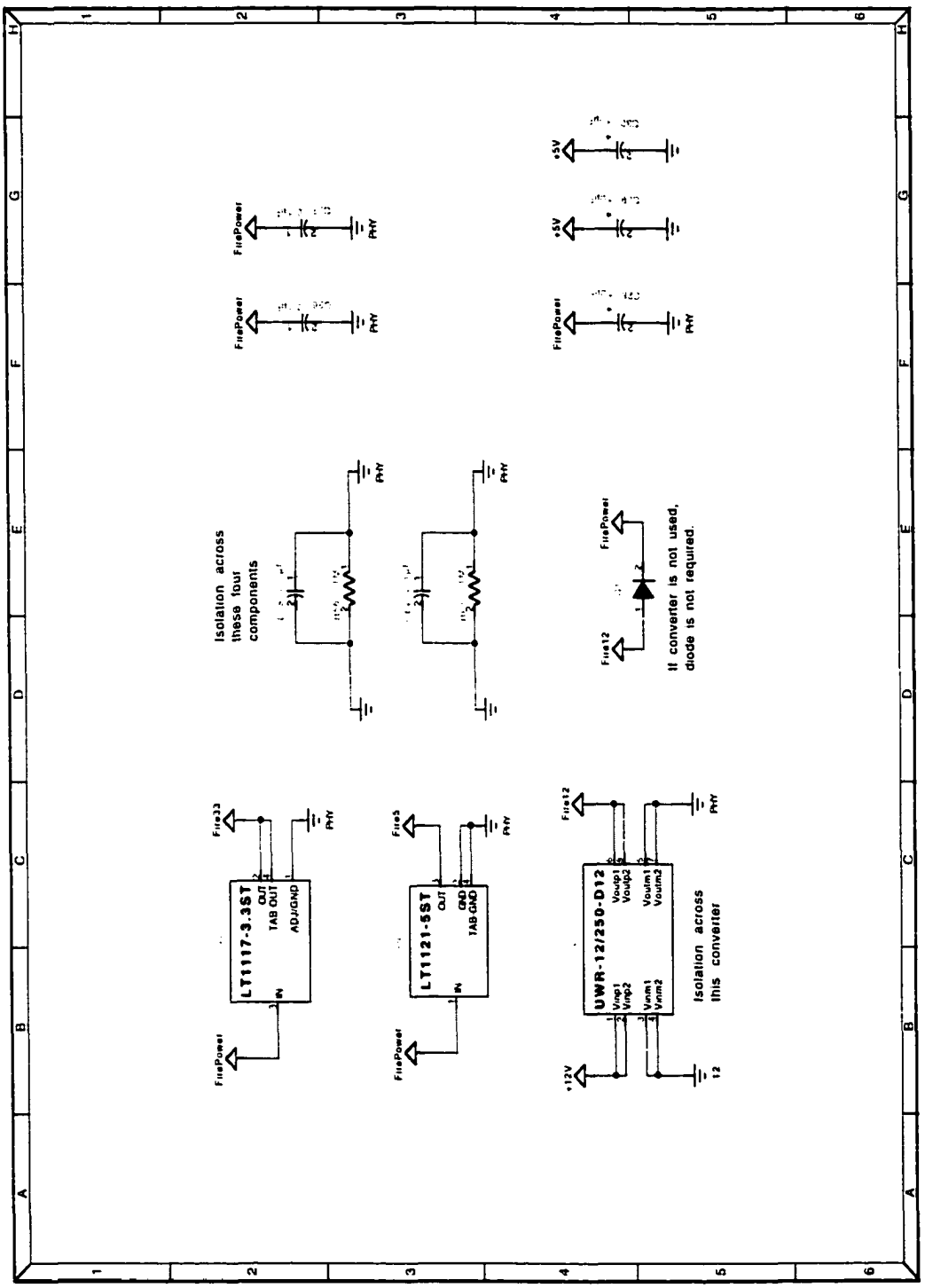
C.1.3.3 Page 3



C.1.3.4 Page 4



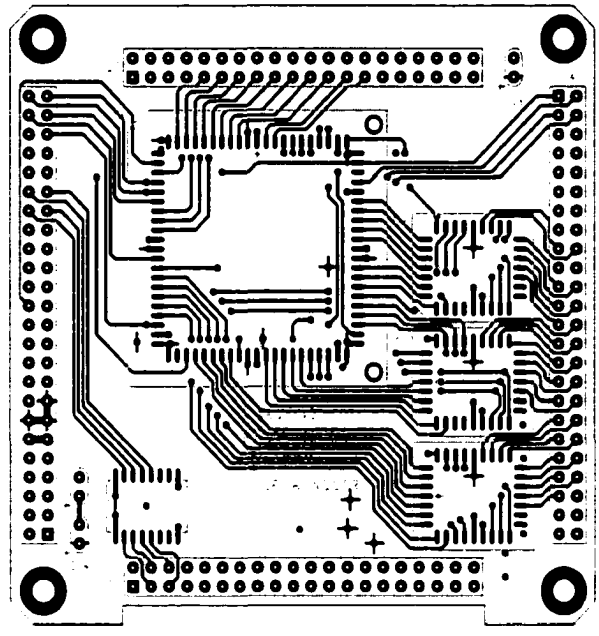
C.1.3.5 Page 5



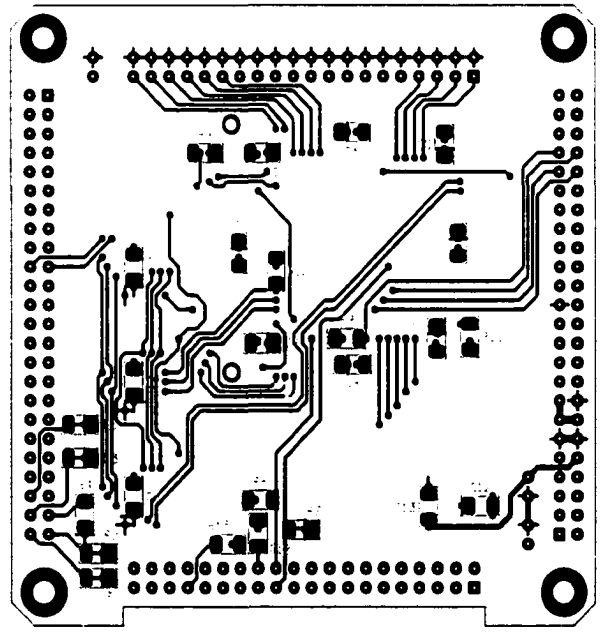
C.2 PCB Layouts

C.2.1 FPGA Board

C.2.1.1 Component Side

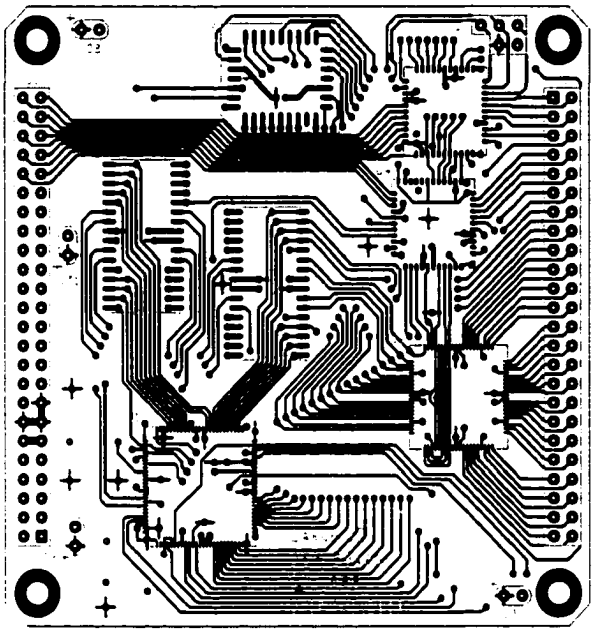


C.2.1.2 Solder Side

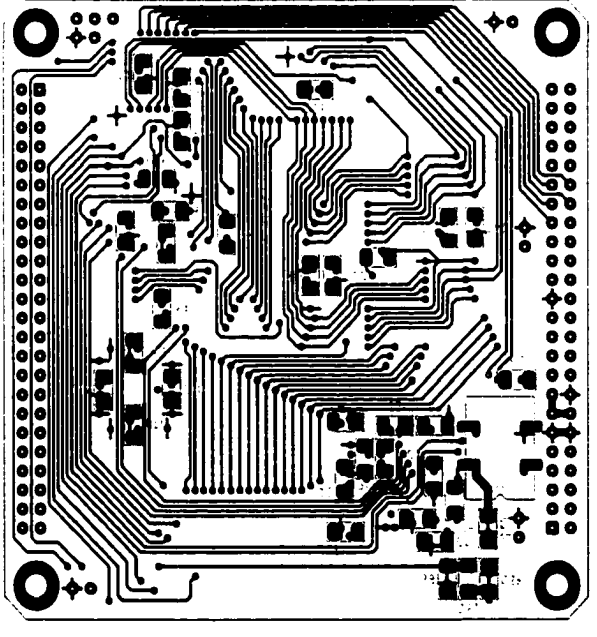


C.2.2 DSP Board

C.2.2.1 Component Side

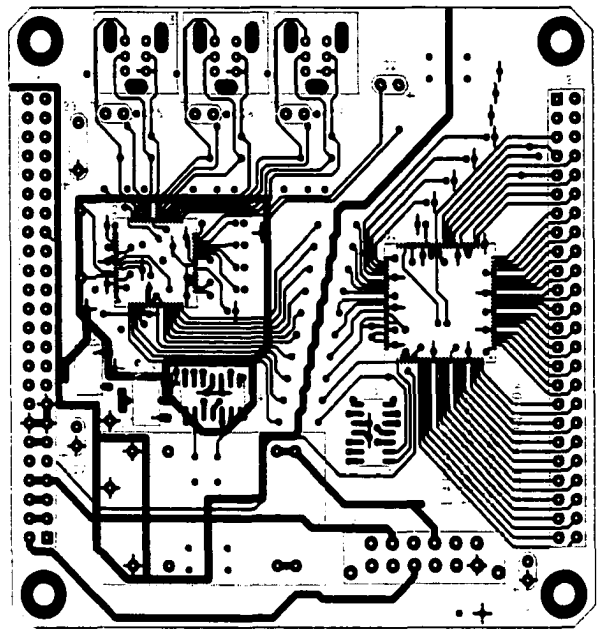


C.2.2.2 Solder Side

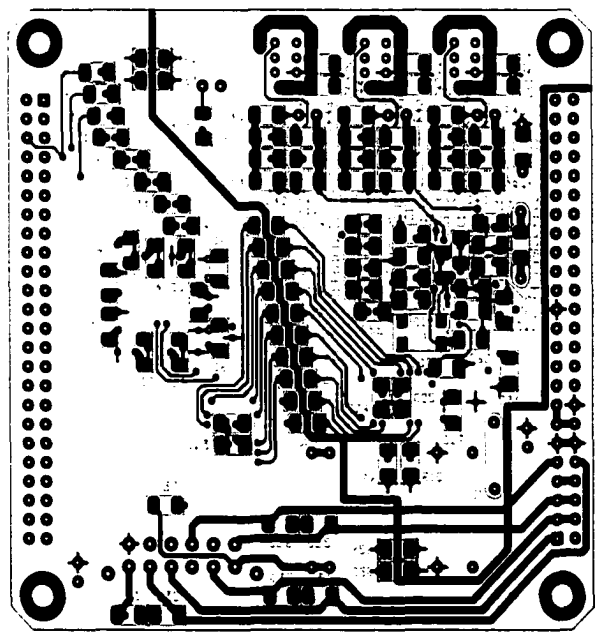


C.2.3 1394 Board

C.2.3.1 Component Side



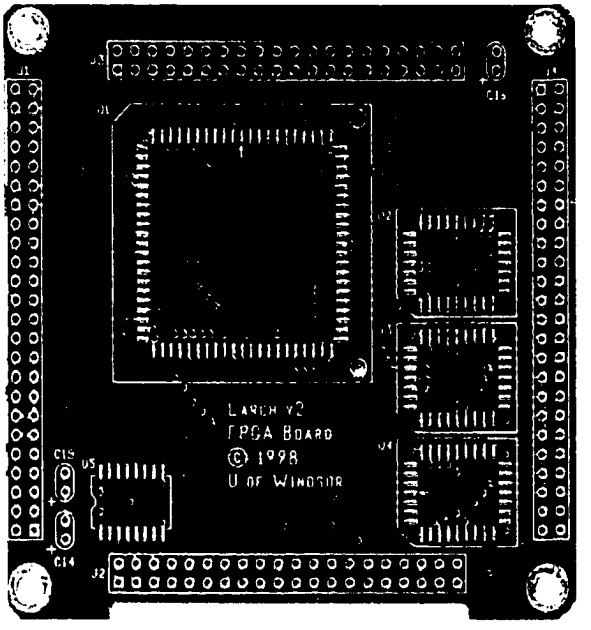
C.2.3.2 Solder Side



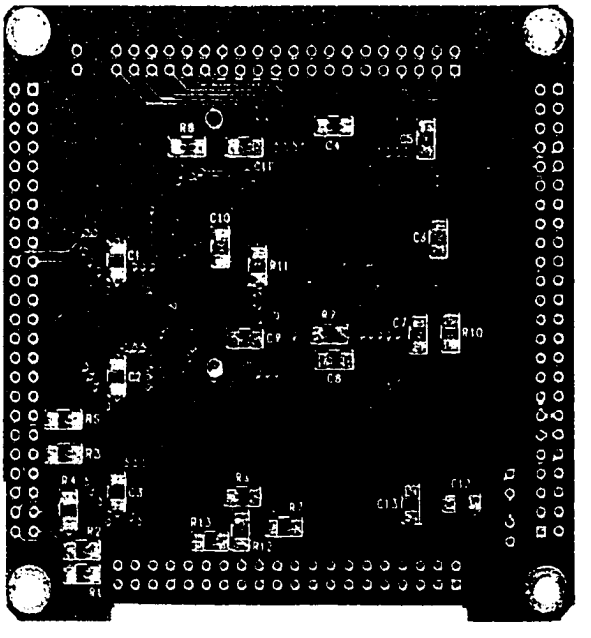
C.3 Fabricated Boards

C.3.1 FPGA Board

C.3.1.1 Component Side

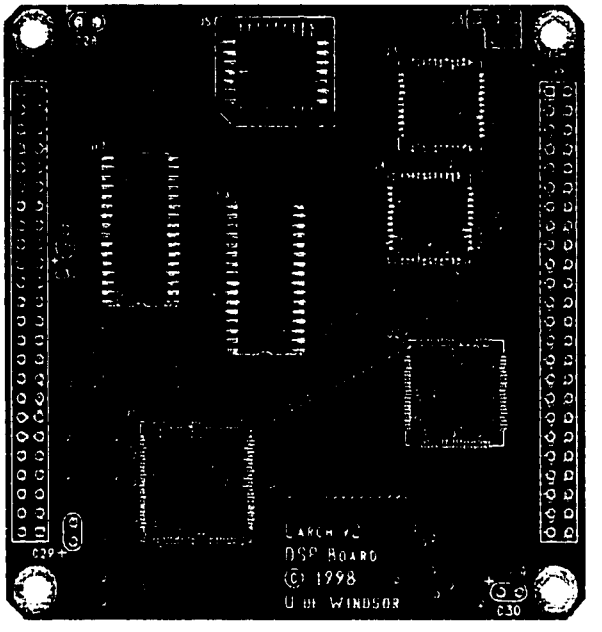


C.3.1.2 Solder Side

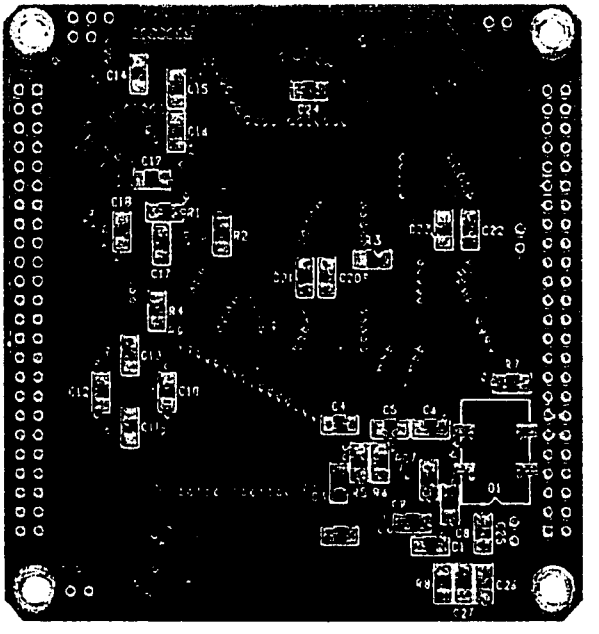


C.3.2 DSP Board

C.3.2.1 Component Side

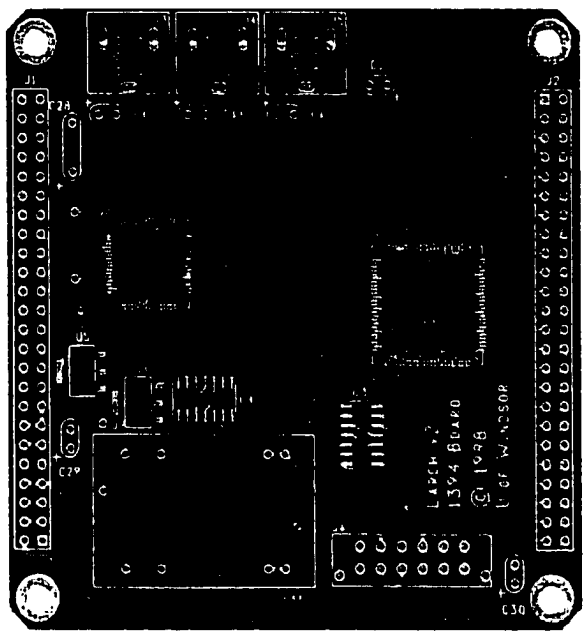


C.3.2.2 Solder Side

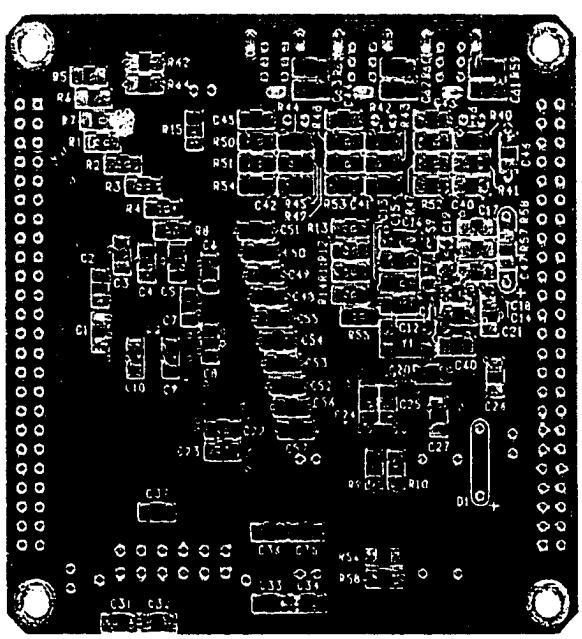


C.3.3 1394 Board

C.3.3.1 Component Side



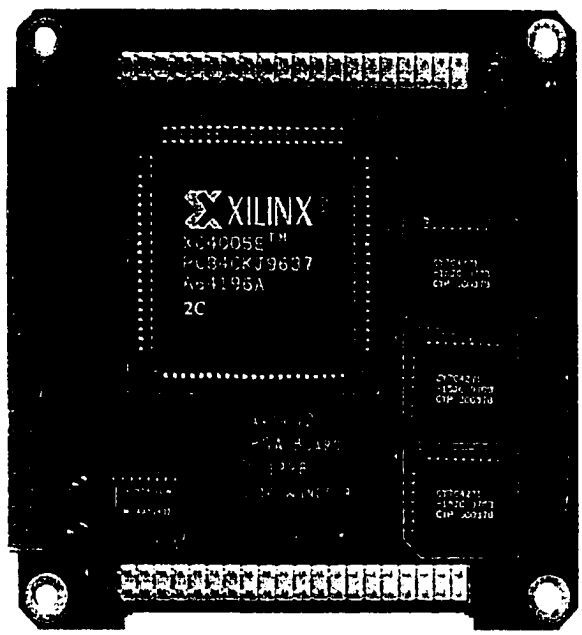
C.3.3.2 Solder Side



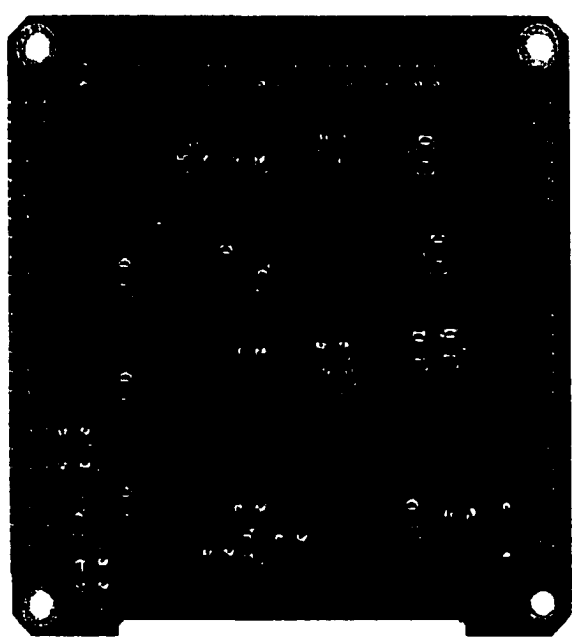
C.4 Assembled Boards

C.4.1 FPGA Board

C.4.1.1 Component Side

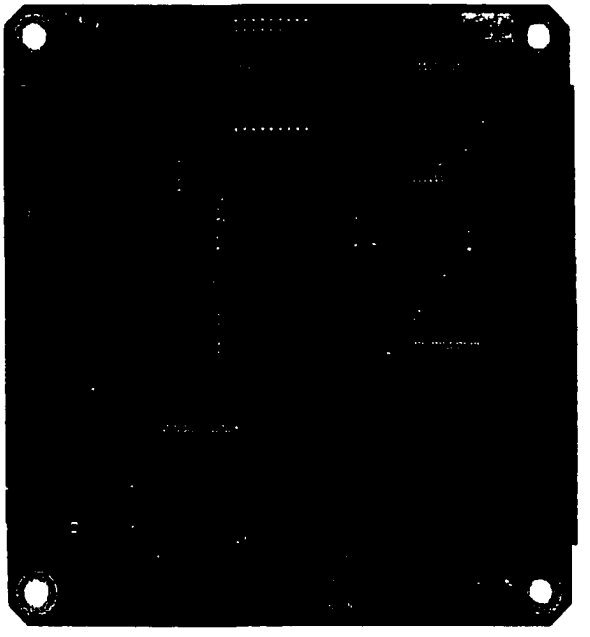


C.4.1.2 Solder Side

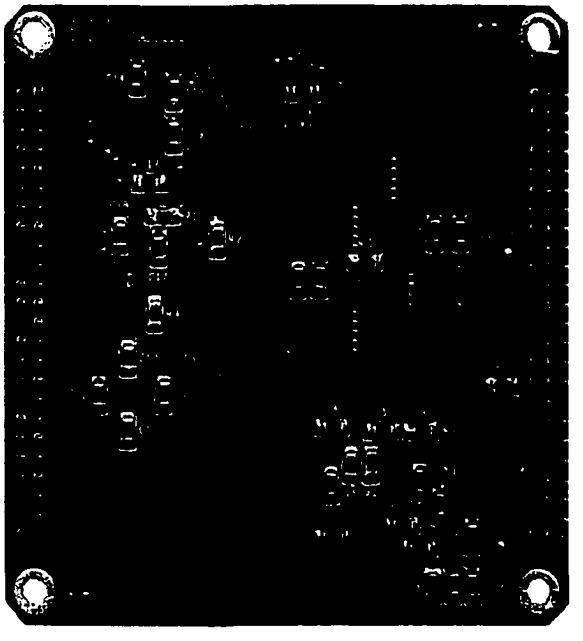


C.4.2 DSP Board

C.4.2.1 Component Side

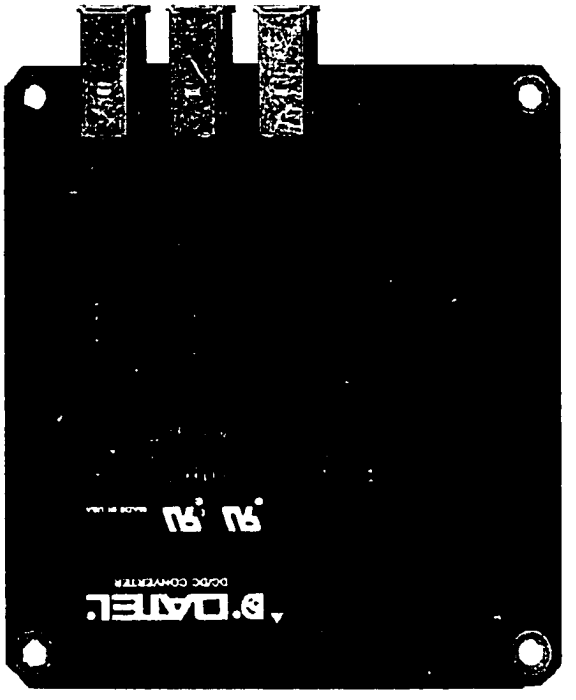


C.4.2.2 Solder Side

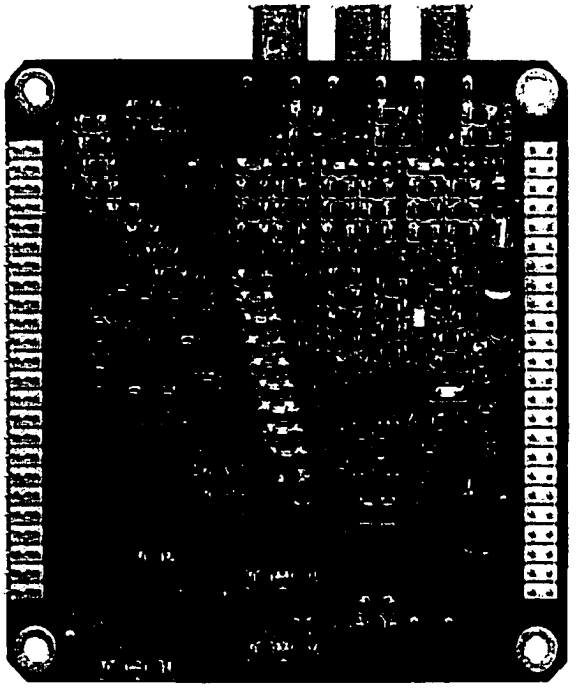


C.4.3 1394 Board

C.4.3.1 Component Side



C.4.3.2 Solder Side



Appendix D

Second Generation System Software Code

D.1 DSP Code

D.1.1 Firmware Loader

To assembler the Firmware Loader, the makeload.cmd command is executed. This command is a simple Windows NT batch file which assembles, links and exports the code to an s19 (Motorola standard) file. The rules for linking (program and data memory base) are in linkload.cmd (not a Windows NT file, but the extension must be.cmd) and the hex output options are in hexload.cmd (ignores any data sections). An external program is written in C which converts the S19 file format into a binary format which can be loaded by an memory programmer or emulator.

D.1.1.1 makeload.cmd

```
..\tms320\dspa loader.asm loader.obj loader.lst -v50
@if errorlevel 1 goto stop
..\tms320\dsplnk linkload.cmd
@if errorlevel 1 goto stop
..\tms320\dsphex hexload.cmd
@if errorlevel 1 goto stop
..\bin\s19tobin\Debug\s19tobin loader
:stop
```

D.1.1.2 linkload.cmd

```
loader.obj
-o loader.out -m loader.map
```

```
MEMORY
{
    PAGE 0 : PROG    : origin = 0FC00h, length = 003F0h
    PAGE 1 : DATA    : origin = 00060h, length = 00020h
}

SECTIONS
{
    .text : load = PROG PAGE 0
    .bss  : load = DATA PAGE 1
}
```

D.1.1.3 hexload.cmd

```
loader.out
-map loader.mxp
-m
-memwidth 16
-order ms

ROMS
{
    PAGE 0:
       PROG:   origin=08000h, len=07fffh, romwidth=16
           files = { loader.s19 }

    PAGE 1:
       DATA:   origin=00000h, len=0ffffh, romwidth=16
           files = { loader.hex }
}

SECTIONS
{
    .text: BOOT
    .bss:
}
```

D.1.1.4 loader.asm

```
; Use register assignments
    .mmregs

; Setup a macro to get a word (16-bits) from the EEPROM device from the last
;  location read
; The device is only 8 bits, therefore 2 read are necessary.

getword.macro

    mar *,ar1           ; set arp to ar1
    lacc*+,8            ; load high byte in accumulator (with 8 bits shift)
    lar ar3,*+,ar2      ; load low byte into ar3, set arp to ar2
    apl #0ffh,ar3       ; set upper 8 bits of ar3 to 0
    or  ar3             ; or ar3 with accumulator

    .endm

getbyte.macro
```

```
    mar *,ar1           ; set arp to ar1
    lacl*+              ; load byte in accumulator
    and #0ffh           ; set upper 8 bits of accumulator to 0

    .endm

    .bssmode,1,0
    .bssword,1,0

    .text

loadsplk#080h,greg      ; set global data access to external from $8000-$FFFF
    ldp #0              ; data page = 0

; ar1 is set to the last memory position used by the internal boot loader
; data for this loader is located right after it in firmware memory

maingetbyte                 ; get mode byte

    saclmode            ; store it

    xor #01h            ; xor it by 1, test if 0
    bcndtryd,neq

    getword                 ; mode 1: prog download, get location word first
    saclar2

    getword                 ; get size, subtract 1
    sub #1
    saclar4

proggetword                 ; get program word
    saclword            ; use indirect program storage opcode tblw
    laclar2
    tblwword
    mar *+,ar4          ; increment write pointer
    banzprog            ; loop until complete

    b   main            ; get another mode

trydlaclmode            ; try mode 2
    xor #02h            ; xor by 2, test if 0
    bcndtryr,neq

    getword                 ; mode 2: data download, get location word first
    saclar2

    getword                 ; set size, subtract 1
    sub #1
    saclar4

datagetword                 ; get data word
    sacl*+,ar4          ; store it, increment write pointer
    banzdata            ; loop until complete

    b   main            ; get another mode

tryrlaclmode            ; type mode 3
    xor #03h            ; xor by 3, test if 0
```

```
    bcnd0,neq           ; if not mode 3, no other modes available, reset system
    getword                 ; mode 3: execute at next word
    splk#0,greg         ; turn off global memory
    bacc                    ; branch to location specified in accumulator
        .end
```

D.1.1.5 ..\bin\s19tobin\main.c

Code for ..\bin\s19tobin\Debug\s19tobin executable.

```
/* Convert S19 file into straight binary file */
/* Only process S2 entries */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Convert a hex string (with size) into a value */
int getval(char *word,int chars)
{
    char temp[80];
    int a;

    /* Copy specified amount */
    strncpy(temp,word,chars);
    /* Terminate it */
    temp[chars]=0;

    /* Use SCANF to convert ascii hex to binary */
    sscanf(temp,"%x",&a);

    return a;
}

/* Write a word to a file stream with high byte first */
void writeword(int word,FILE *stream)
{
    fputc((word>>8) & 255,stream);
    fputc(word & 255,stream);

    return;
}

/* Write a byte to a file stream */
void writebyte(int byte,FILE *stream)
{
    fputc(byte & 255,stream);

    return;
}
```

```
/* Main code */
/* Usage: s19tobin [filename] */
int main (int argc, char *argv[])
{
    FILE *in, *out;
    char inname[100],outname[100];
    char line[80];
    unsigned int sum;
    int point,start=0xffffff,value,base=-1,i,j;

    /* exist if arguments not correct */
    if (argc!=2 || argv[1]==NULL) return 1;

    /* derive input and output file names */
    sprintf(inname,"%s.s19",argv[1]);
    sprintf(outname,"%s.bin",argv[1]);

    /* open input file in text mode, exit if error */
    in = fopen(inname,"rt");
    if (in==NULL) return 2;

    /* open ouput file in binary mode, exit if error */
    out = fopen(outname,"wb");
    if (out==NULL) return 2;

    /* Ignore first line; we know it is an s0 entry */
    fgets(line,79,in);

    /* loop until no lines left */
    while (fgets(line,79,in))
    {
        /* set check sum to 0 */
        sum=0;
        /* set counter to 0 */
        point=0;

        /* loop until end of line */
        while (point<(int)strlen(line))
        {
            /* Check first character */
            switch(line[point])
            {
                /* All valid lines begin with S */
                case 'S':
                    point ++;
                    switch(line[point])
                    {
                        /* Entry 2 */
                        case '2':
                            point ++;
                            /* Read in values remaining */
                            i = getval(line+point,2)-4;
                            point += 2;
                            point += 2;
                            /* Get memory address */
                            value = getval(line+point,4);
                            /* Set start of memory if necessary */
                            if (start>value) start=value;
                            if (value>base) base=value;
```

```
                        point += 4;
                        /* Read program values and store in binary file*/
                        for (j=0;j<i;j++)
                        {
                            value = getval(line+point,2);
                            writebyte(value,out);
                            point += 2;
                            base++;
                        }
                        /* Exit this line */
                        point += 10;
                        break;

                    default:
                        /* Any other entry than 2, exit line */
                        point = 100;
                        break;
                }
                break;

            default:
                /* Not an S command, exit */
                return 5;

        }
    }
}

/* Close files */
fclose(in);
fclose(out);

/* Output binary size */
printf("Ok: %x - %x\n",start,base);

return 0;
}
```

D.1.2 Firmware Builder

As with the loader, another Window NT script is written (makecode.cmd) which will assembler all the files (will exit if any error occurs). The linkcode.cmd contains the linker information for all the sections of code. The hexcode.cmd contains information for both data and program memory. A new convert is written (s19tomod) to change the S19 file into a format which the firmware loader understands.

D.1.2.1 makecode.cmd

```
..\tms320\dspa main.asm main.obj main.lst -v50 -x
@if errorlevel 1 goto stop
..\tms320\dspa fpga.asm fpga.obj fpga.lst -v50 -x -w
@if errorlevel 1 goto stop
```

```
..\tms320\dspa rs232.asm rs232.obj rs232.lst -v50 -x -w
@if errorlevel 1 goto stop
..\tms320\dspa tsb.asm tsb.obj tsb.lst -v50 -x -w
@if errorlevel 1 goto stop
..\tms320\dspa debug.asm debug.obj debug.lst -v50 -x -w
@if errorlevel 1 goto stop
..\tms320\dsplnk linkcode.cmd
@if errorlevel 1 goto stop
..\tms320\dsphex hexcode.cmd
@if errorlevel 1 goto stop
..\bin\s19tomod\Debug\s19tomod prog 1
:stop
```

D.1.2.2 linkcode.cmd

```
main.obj fpga.obj rs232.obj tsb.obj debug.obj
-o firmware.out -m firmware.map

MEMORY
{
    PAGE 0 : PROG    : origin = 01000h, length = 0f000h
    PAGE 1 : DATA    : origin = 00800h, length = 0f800h
    PAGE 1 : PAGE0   : origin = 00070h, length = 00008h
    PAGE 1 : ONCHIP : origin = 00300h, length = 00200h
    PAGE 2 : EEPROM : origin = 08000h, length = 07ff0h
}

SECTIONS
{
    .text : load = PROG PAGE 0
    command : load = ONCHIP PAGE 1
    grfdata : load = ONCHIP PAGE 1
    fpga : load = ONCHIP PAGE 1
    debug : load = ONCHIP PAGE 1
    rsdata : load = ONCHIP ALIGN(128) PAGE 1
    grfpack : load = DATA ALIGN(256) PAGE 1
    grftran : load = DATA ALIGN(256) PAGE 1
    grffifo : load = DATA ALIGN(256) PAGE 1
    sendbuf : load = DATA ALIGN(256) PAGE 1
    buffers : load = DATA ALIGN(256) PAGE 1
    .data : load = DATA ALIGN(256) PAGE 1
}
```

D.1.2.3 hexcode.cmd

```
firmware.out
-map firmware.mxp
-m
-memwidth 16
-order ms

ROMS
{
    PAGE 0:
        PROG:   origin=00000h, len=0ffffh, romwidth=16
            files = { prog.s19 }

    PAGE 1:
```

```
        DATA:   origin=00000h, len=0ffffh, romwidth=16
            files = { data.s19 }
}
```

D.1.2.4 ..\bin\s19tomod\main.c

Code for ..\bin\s19tomod\Debug\s19tomod executable.

```
/* Convert S19 into special firmware loader format */
/* Only process S2 entries */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Convert a hex string (with size) into a value */
int getval(char *word,int chars)
{
    char temp[80];
    int a;

    /* Copy specified amount */
    strncpy(temp,word,chars);
    /* Terminate it */
    temp[chars]=0;

    /* Use SCANF to convert ascii hex to binary */
    sscanf(temp,"%x",&a);

    return a;
}

/* Write a word to a file stream with high byte first */
void writeword(int word,FILE *stream)
{
    fputc((word>>8) & 255,stream);
    fputc(word & 255,stream);

    return;
}

/* Write a byte to a file stream */
void writebyte(int byte,FILE *stream)
{
    fputc(byte & 255,stream);

    return;
}

/* Main code */
/* Usage: s19tomod [filename] [mode] */
int main (int argc, char *argv[])
{
    FILE *in, *out;
    char inname[100],outname[100];
```

```
char line[80];
unsigned int sum;
int point,start=-1,value,base=-1,i,j,mode,bufpnt;
unsigned int buffer[32768];

/* exist if arguments not correct */
if (argc!=3 || argv[1]==NULL || argv[2]==NULL) return 1;

/* derive input and output file names */
sprintf(inname,"%s.s19",argv[1]);
sprintf(outname,"%s.bin",argv[1]);

/* open input file in text mode, exit if error */
in = fopen(inname,"rt");
if (in==NULL) return 2;

/* open ouput file in binary mode, exit if error */
out = fopen(outname,"wb");
if (out==NULL) return 2;

/* Get mode entry */
sscanf(argv[2],"%d",&mode);

/* Ignore first line; we know it is an s0 entry */
fgets(line,79,in);

/* loop until no lines left */
while (fgets(line,79,in))
{
    /* set check sum to 0 */
    sum=0;
    /* set counter to 0 */
    point=0;

    /* loop until end of line */
    while (point<(int)strlen(line))
    {
        /* Check first character */
        switch(line[point])
        {
            /* All valid lines begin with S */
            case 'S':
                point ++;
                switch(line[point])
                {
                    /* Entry 2 */
                    case '2':
                        point ++;
                        /* Read in values remaining */
                        i = getval(line+point,2)-4;
                        point += 2;
                        point += 2;
                        /* Get memory address */
                        value = getval(line+point,4);
                        point += 4;
                        if (start<0)
                        {
                            /* First memory address ever */
                            start=value;
```

```
                        bufpnt=0;
                        base=0;
                    }
                    /* Check if last address (on previous line) */
                    /* is equal to new address */
                    if (value!=(start+(base/2)))
                    {
                        /* flush out existing data */
                        /* mode */
                        fputc(mode,out);
                        /* location */
                        writeword(start,out);
                        /* size */
                        writeword(bufpnt/2,out);
                        /* data or program memory */
                        for(j=0;j<bufpnt;j++)
                        {
                            writebyte(buffer[j],out);
                        }
                        /* Output size to user */
                        printf("Ok: %x - %x\n",start,start+bufpnt/2);
                        start=value;
                        bufpnt=0;
                    }
                    /* if the same, record ascii data into buffer */
                    for (j=0;j<i;j++)
                    {
                        value = getval(line+point,2);
                        buffer[bufpnt++]=value;
                        point += 2;
                        base++;
                    }
                    /* Exit this line */
                    point += 10;
                    break;

                default:
                    /* Any other entry than 2, exit line */
                    point = 100;
                    break;
            }
            break;

        default:
            /* Not an S command, exit */
            return 5;

    }
  }
}

/* Input data done, flush buffer if necessary */
if (bufpnt!=0)
{
    /* mode */
    fputc(mode,out);
    /* location */
    writeword(start,out);
    /* size */
```

```
        writeword(bufpnt/2,out);
        /* data or program memory */
        for(j=0;j<bufpnt;j++)
        {
            writebyte(buffer[j],out);
        }
        /* Output size to user */
        printf("Ok: %x - %x\n",start,start+bufpnt/2);
        start=value;
        bufpnt=0;
    }

    /* Close files */
    fclose(in);
    fclose(out);

    /* Everything OK */
    printf("Ok.\n");

    return 0;
}
```

D.1.3 Firmware Code

D.1.3.1 main.inc

```
; Set global routines for MAIN module
    .global commandl,commandh,hostnode,hostfifo
    .global hostentl,hostenth,hostmask,infpga
    .global fpksizeb,fpksizew,fpksizeq
```

D.1.3.2 debug.inc

```
; Set global routines for debug module
    .global debugg,debuga,debugas,debugar,debugr,debugs,debugws,debugwr
```

D.1.3.3 rs232.inc

```
; Set global routines for RS232 module
    .global rsinit,rsdump,rssend,rsstat
```

D.1.3.4 fpga.inc

```
; Set global routines for FPGA module
    .global fpgainit,fpgald,fpgaput,fpgaget,fpgadump,fpgarec
```

D.1.3.5 tsb.inc

```
; Set global routines for 1394 module
     .global tsbinit,tsbdump,tsbfifo,tsbfifoi
```

D.1.3.6 main.asm

```
; Use register assignments
    .mmregs
```

```
; Include global references
    .include "main.inc"
    .include "fpga.inc"
    .include "rs232.inc"
    .include "debug.inc"
    .include "tsb.inc"

; Location of data from FPGA and size
infpga.usect "buffers",256,0
fpksizeb.set504
fpksizew.set 252
fpksizeq.set126

; Variables
commandl.usect "command",1,0
commandh.usect "command",1,0
hostnode.usect "command",1,0
hostfifo.usect "command",1,0
hostentl.usect "command",1,0
hostenth.usect "command",1,0
hostmask.usect "command",1,0

result.usect "command",1,0
skip.usect "command",1,0
lastcap.usect "command",1,0

templ.usect "command",1,0
temp2.usect "command",1,0
tcount.usect "command",1,0
tprint.usect "command",1,0

; Setup COMMAND data page
compage.set    commandl

temp.set    061h

    .text

; Initialize DSP
startsetc    intm
    ldp     #0
    clrc    sxm
    clrc    ovm

; Disable wait-states
    lacl    #000h
    sacl    pdwsr
    sacl    iowsr
    sacl    greg

    lacl    cwsr
    and     #0ffe0h
    or      #00010h
    sacl    cwsr

; Print a period
    lacl    #02eh
    call    debuga
```

```
; Initialize all sub-systems
    call    tsbinit
    call    tsbfifoi
    call    rsinit
    call    fpgainit
    call    tsbfifoi

; Initialize local and global variables
    ldp     #compage

    setc    intm
    lacl    #0
    sacl    commandl
    sacl    commandh
    clrc    intm

    sacl    skip
    sacl    lastcap
    sacl    tcount
    sacl    tprint

    lacl    #1
    sacl    hostmask

; Main routine
main
    ldp     #compage

; Reset if asked
    bit     commandl,15   ;reset (0)
    bcnd    0,tc

; Send and Received and RS232 packet to and the camera
    bit     commandl,14   ;camera rs232 (3,2,1)
    bcnd    main0,ntc

; Send packet
    call    rssend

; Wait for it
$1
    call    rsstat
    bcnd    $1,eq

; Save status
    ldp     #compage

    sfl     ;shift 2
    sfl

    sacb

    setc    intm

    lacl    commandl
    orb
    and     #0fffdh
    sacl    commandl
```

```
    clrc    intm

; Output packet to terminal if debugging
;   call    rsdump

    b       main

main0
; Program the FPGA
    bit     command1,11    ;fpga (5,4)
    bcnd    main1,ntc

; Program it
    call    fpgald

; Save status
    ldp     #compage

    rpt     #4    ;shift 5
    sfl

    sacb

    setc    intm

    lacl    command1
    orb
    and     #0ffefh
    sacl    command1

    clrc    intm

    b       main

main1
; Write a parameter to the FPGA
    bit     command1,9     ;fpga (6)
    bcnd    main2,ntc

; Debug, output stream
;   lacl    commandh
;   and     #01fffh
;   call    debugwr

; Send parameter
    ldp     #compage
    lacl    commandh
    and     #01fffh

    call    fpgaput

; Get parameter (loop-back)
    call    fpgaget

;   and     #00ffh

; Save status
    ldp     #compage
```

```
    setc    intm

;   sacl    commandh
;   call    debugwr

    ldp     #compage
    lacl    commandl
    and     #0ffbfh
    sacl    commandl

    clrc    intm

    b       main

main2
; Enable/disable capture
    bit     commandl,8      ;capture (7)
    bcnd    main2z,ntc

    lacl    lastcap
    bcnd    main2c,neq

; Beginning of capture, print @'s until data comes in
    lacl    #040h
    call    debuga

; Initalize the FIFO
    call    tsbfifoi

    ldp     #compage
    lacl    #0
    sacl    skip

main2c
; If failed transmittin last section of data, try sending it again
    lacl    skip
    bcnd    main2a,neq

; Get data from camera
    call    fpgarec
    bcnd    main3,neq

main2a
; Send data to HOST
    lar     ar2,#infpga
    call    tsbfifo
    bcnd    main2b,eq

    lacl    #1
main2b
    ldp     #compage
    sacl    skip

; Indicate we have sucessfully got data from the camera
    lacl    #1
    sacl    lastcap
    b       main3
```

```
main2z
    lacl    #0
    sacl    lastcap

main3

main9

; Debug ans TSB transactions
;   call    tsbdump

; Get a character from the terminal
    call    debugg
    bcnd    main,eq

    call    debuga

    sacb

; Reset system if it's a SPACE
    xor     #020h
    bcnd    0,eq

    b       main

        .end
```

D.1.3.7 debug.asm

```
; Use register assignments
    .mmregs

; Include global references
    .include "debug.inc"

; Setup temporary system register storage */
debugst0.set    07eh
debugst1.set    07fh

; Setup debug variables in debug section
debugacc.usect  "debug",1,0
debuggrg.usect  "debug",1,0
debugdws.usect  "debug",1,0
debugsb.usect   "debug",1,0
debugext.usect  "debug",1,0
debugscr.usect  "debug",1,0
debugchr.usect  "debug",1,0
debugar4.usect  "debug",1,0

; Set debug data page to first variable used
debugpage .set debugacc

    .text

; SAVENV macro saves all system registers and data page into temporary variables
saveenv.macro

    sst     #0,debugst0     ; Save system registers
    sst     #1,debugst1
```

```
    ldp     #debugpage      ; Set data page to debug page

    sacl    debugacc        ; Save accumulator

    clrc    sxm             ; Disable sign extension mode

; Save system variables (removed)
;   lacl    greg
;   sacl    debuggrg
;   lacl    pdwsr
;   sacl    debugdws
;   lacl    #080h
;   sacl    greg
;   lacc    #0ff00h
;   sacl    pdwsr

    .endm

; RESTENV macro restores all system registers and data page from temporary
variable
restenv  .macro

    ldp     #debugpage

; Restore system variable (removed)
;   lacl    debuggrg
;   sacl    greg
;   lacl    debugdws
;   sacl    pdwsr

    lacl    debugacc        ; Restore accumulator

    ldp     #0              ; reset data page

    lst     #0,debugst0     ; restore system registers
    lst     #1,debugst1

    .endm

; get a byte from the emulator terminal
debuggsaveenv                ; save environment

    call    igb             ; get bytes

    sacl    debugacc        ; store it in restore registers

    restenv                 ; restore environment

    ret                     ; leave

; Print accumulator (ascii) and space to terminal
debugassaveenv               ; save environment

    lacl    #020h           ; get ready to print a space
    b       debuga0         ; print saved accumulator, space, restore environment,
leave

; Print accumulator (ascii) and CR to terminal
```

```
debugarsaveenv              ; save environment

   lacl    #00dh          ; get ready to print a CR
   b       debuga0        ; print saved accumulator, CR, restore environment,
leave

; Print accumulator (ascii) to terminal
debugasaveenv               ; save environment

   lacl    #0             ; set second byte to send to NULL
debuga0sacl    debugext

   lacl    debugacc       ; send saved environment accumulator
   call    isb            ; send to terminal

   lacl    debugext       ; send second byte if not NULL
   bcnd    debuga1,eq
debuga2call    isb          ; send to terminal

debuga1restenv              ; restore environment
   ret                     ; leave

; Send a CR to the terminal
debugrsaveenv               ; save environment

   lacl    #00dh          ; get ready to print a CR
   b       debuga2        ; continue to above routine

; Send a SPACE to the terminal
debugssaveenv               ; save environment

   lacl    #020h          ; get ready to print a space
   b       debuga2        ; continue to above routine

; Convert a 16bit value to ascii hex with a CR to the terminal
debugwrsaveenv              ; save environment

   lacl    #00dh          ; get ready to print a CR
   b       debugws0       ; continue to next routine

; Convert a 16bit value to ascii hex with a space to the terminal
debugwssaveenv              ; save environment

   lacl    #020h          ; get ready to print a space
debugws0sacl    debugext ; set last byte to send

   sar     ar4,debugar4   ; save ar4

   lacl    debugacc       ; get stored accumulator
   sacl    debugscr

   lar     ar4,#3         ; set counter to 3
debugw0lacc    debugscr,4    ; shift 4 bits
   sacl    debugscr
   sach    debugchr       ; save upper 1 char
   lacl    debugchr       ; load char
   and     #00fh          ; Convert 0-9 and A-F to ASCII
   setc    c
   sub     #10            ; Subtract 10
```

```
    bcnd    debugw1,lt    ;
    add     #7            ; If greater than equal to 0, add 7
debugwladd     #03ah          ; Add 57
    call    isb           ; xmit the ASCII character
    mar     *,ar4
    banz    debugw0,*-    ; do the rest of the nybles

    lacl    debugext      ; print last character (space or CR)
    call    isb

    lar     ar4,debugar4  ; restore ar4

    restenv               ; restore environment
    ret                   ; leave

; Send a byte to the terminal
;  Environment must be changed!
isb sacl    debugsb       ; save outgoing character

    lamm    greg          ; save global register
    sacl    debuggrg

    lamm    pdwsr         ; save waitsate register
    sacl    debugdws

    lacl    #080h         ; set global memory
    samm    greg

    lacc    #0ff00h       ; set external waitstates to 7 for emulator
    samm    pdwsr

    ldp     #511          ; set data page to 511 = $FF80

isb0rpt     #255          ; wait a while
    nop
    lacl    070h          ; read from $FFF0; arbitrate with emulator
    rpt     #252          ; wait a while
    nop
    lacl    071h          ; read from $FFF1
    and     #0ffh         ; set upper 8 bits to 0!
    bcnd    isb0,neq      ; loop back if not 0 (data still transmitting to
terminal)

    ldp     #debugpage    ; set back to debugger page
    lacl    debugsb       ; get outgoing data

    ldp     #511          ; set to emulator page
    sacl    070h          ; save value to $FFF0
    lacl    #1            ; set 1 to $FFF1 to indicate write
    sacl    071h

    lacl    072h          ; arbitrate with emulator

    ldp     #debugpage    ; set back to debugger page

    lacl    debuggrg      ; restore environment
    samm    greg

    lacl    debugdws
```

```
    samm    pdwsr

    lacl    debugsb         ; restore accumulator

    ret                     ; return

; Get a byte to the terminal
;  Environment must be changed!
igb lacl    #0              ; reset incoming data
    sacl    debugsb

    lamm    greg            ; save global register
    sacl    debuggrg

    lamm    pdwsr           ; save waitsate register
    sacl    debugdws

    lacl    #080h           ; set global memory
    samm    greg

    lacc    #0ff00h         ; set external waitstates to 7 for emulator
    samm    pdwsr

    ldp     #511            ; set data page to 511 = $FF80
    lacl    072h            ; read from $FFF2
    and     #000ffh
    bcnd    igb1,eq         ; if zero, no chars available from terminal, leave

    lacl    073h            ; read from $FFF3
    and     #000ffh         ; set upper 8 bits to 0

    ldp     #debugpage      ; set back to debugger page
    sacl    debugsb         ; store in incoming data register

    ldp     #511            ; set back to emulator page
    lacl    #0
    sacl    072h            ; set $FFF2 and $FFF3 to zero to indicate we got it
    sacl    073h

igb1
    ldp     #debugpage      ; set back to debugger page

    lacl    debuggrg        ; restore environment
    samm    greg

    lacl    debugdws
    samm    pdwsr

    lacl    debugsb         ; restore accumulator

    ret                     ; return

    .end
```

D.1.3.8 rs232.asm

```
; Use register assignments
    .mmregs
```

```
; Include global references
    .include "rs232.inc"
    .include "debug.inc"

rssize.usect "rsdata",1,0
rsbufin.usect "rsdata",29,0
rsosize.usect "rsdata",1,0
rsbufout.usect "rsdata",29,0
rsbufpnt.usect "rsdata",1,0
rsobufpnt.usect "rsdata",1,0
rspdwsr.usect "rsdata",1,0
rsgreg.usect "rsdata",1,0
rsar2.usect "rsdata",1,0
rsar3.usect "rsdata",1,0
rsar4.usect "rsdata",1,0
rsar5.usect "rsdata",1,0
rsrev.usect "rsdata",1,0
rscount.usect "rsdata",1,0
rscur.usect "rsdata",1,0
rspackok.usect "rsdata",1,0
rspackbad.usect "rsdata",1,0
rschksum.usect "rsdata",1,0
rsochksum.usect "rsdata",1,0
rstemp.usect "rsdata",1,0
rstemp2.usect "rsdata",1,0

; set RS232 page
rspage.set   rssize

; reverse bits in accumulator for RS232 transmission (with table)
;  DSP serial port is MSB, RS232 is LSB
revbits.macro

    .newblock           ; reset local branches

    and     #000ffh     ; set upper 8 bits to 0
    sfr                 ; shift right 1 bit
    bcnd    $1,c        ; was value even
    add     #rs232tr    ; yes, get value from table
    tblr    rsrev
    lacl    rsrev
    bsar    8           ; shift right 8 bits (use upper bits)
    b       $2

$1  add     #rs232tr    ; no, odd
    tblr    rsrev       ; get value from table (use lower bits)
    lacl    rsrev
$2  and     #000ffh     ; set upper bits to 0

    .newblock

    .endm

; set a byte (in accumulator) to Dalsa MCU
send.macro

    .newblock

    revbits             ; reverse bits
```

```
    samm    dxr             ; store in MSB shift register to CPLDs

$1  lamm    spc             ; wait for transmission to be finished
    and     #00800h
    bcnd    $1,eq

    rpt     #65535          ; wait LONGER...
    nop

    .newblock

    .endm

    .text

; RSINIT: set interrupt, timers and serial registers
;  CPLD handles most of the RS232 formatting of bits
;  CPLD requires a pulse 1/(baud*8) times a second
rsinitsetc    intm          ; disable interrupts

    ldp     #0              ; set data page 0

    lacc    #isr            ; set interrupt handler
    sacl    064h            ; serial communication vector at $64 in ROM version of DSP

    ldp     #rspage         ; set RS232 page

    lacc    #008ech         ; enable 16 bit, frammed mode, reset transmitter/receiver
    samm    spc

    lacc    #0326           ; set timer pulse to 25x10^6 / 9600 / 8
    samm    prd

    lacc    #00020h         ; set timer in continuous mode
    samm    tcr

    lacl    #0              ; reset variables
    sacl    rscount
    sacl    rspackok
    sacl    rspackbad
    sacl    rschksum
    lacc    #rsbufin
    add     #rsbufin
    sacl    rsbufpnt

    lamm    imr             ; set interrupt registers
    or      #00010h
    samm    imr

    lacl    #010h
    samm    ifr

    clrc    xf              ; set RS232 RTS high
    out     rsrev,02004h

    ldp     #0

    clrc    intm            ; enable interrupts
    ret                     ; leave
```

```
; ISR: interrupt service routine for RS232 (receive)
;  Receives bytes and processes DALSA MCU packet
isr ldp     #rspage      ; set page (previous page is saved by hardware)

    lamm    pdwsr        ; interrupt may happen in debugger code, must compensate
    sacl    rspdwsr
    lamm    greg
    sacl    rsgreg

    lacl    #0
    samm    pdwsr
    samm    greg

    sar     ar2,rsar2    ; save ar's to be used
    sar     ar3,rsar3
    sar     ar4,rsar4
    sar     ar5,rsar5

    lacl    rspackok     ; leave if status registers are not cleared
    bcnd    isrleave,neq

    lacl    rspackbad
    bcnd    isrleave,neq

    lamm    drr          ; get received bit
    revbits              ; reverse the order
    sacl    rscur        ; store it

    lacl    rscount      ; check packet size
    sub     #5
    bcnd    isrsg,eq     ; header just completed, record size portion
    bcnd    isrget,gt    ; more than header

    lacl    rscount
    add     #rssync      ; verify header is correct
    tblr    rsrev
    lacl    rscur
    xor     rsrev
    bcnd    isrreset,neq ; if not, reset packet receiver

    b       isrinc

isrsglacl    rscur       ; save size value in header
    add     #4
    sacl    rssize

isrchklacl   rschksum    ; update checksum
    add     rscur
    sacl    rschksum

isrinclacl   rscount     ; increment counter
    add     #1
    sacl    rscount

    b       isrleave     ; leave ISR

isrgetlacl   rscount     ; check if packet complete
    xor     rssize
```

```
    bcnd    isrdone,eq

    lacl    rsbufpnt    ; packet not done, record in buffer
    sfr                 ;  buffer is 16 bit wide, sub-divide into two 8-bits
    sacl    rstemp
    lar     ar2,rstemp
    mar     *,ar2
    lacc    *,8
    or      rscur
    sacl    *
    lacl    rsbufpnt    ; increment buffer counter
    add     #1
    sacl    rsbufpnt
    b       isrchk      ; update checksum

isrdonelacl    rschksum    ; packet done, verify checksum
    xor     #000ffh
    add     #1
    and     #000ffh
    xor     rscur
    bcnd    isrreset,neq ;isrbad ; packet is bad, leave!

    lacl    rsbufpnt    ; NULL terminate buffer
    sfr
    sacl    rstemp
    lar     ar2,rstemp
    mar     *,ar2
    lacc    *,8
    sacl    *
    lacl    rsbufpnt
    add     #1
    sacl    rsbufpnt

    lacl    rssize      ; modify size, remove header
    sub     #6
    sacl    rssize

    lacl    #1          ; set packet OK!
    sacl    rspackok

    b       isrleave    ; leave

isrbadlacl    #1          ; set packet BAD!
    sacl    rspackbad

    b       isrleave    ; leave

isrresetlacl    #0  ;reset receive packet registers
    sacl    rscount
    sacl    rspackok
    sacl    rspackbad
    sacl    rschksum
    lacc    #rsbufin
    add     #rsbufin
    sacl    rsbufpnt

isrleavelar     ar5,rsar5 ; restore ar's and environment
    lar     ar4,rsar4
    lar     ar3,rsar3
```

```
    lar     ar2,rsar2

    lacl    rsgreg
    samm    greg
    lacl    rspdwsr
    samm    pdwsr

    lacl    #010h       ; set interrupt processed bit
    samm    ifr

    rete

; packet header
rssync.word   0ffh,055h,0aah,05ah,0a5h

; RSSEND: format DALSA MCU pakcet and send it.
rssendldp     #rspage     ; set rs232 data page

    lacl    rsosize     ; leave if data is zero size
    bcnd    rssende,eq

    lar     ar2,#rssync ; ready to send header
    lar     ar3,#4
    mar     *,ar2

rssend0sar      ar2,rstemp2 ; send header
    mar     *+,ar3
    lacl    rstemp2
    tblr    rstemp2
    lacl    rstemp2
    send
    banz    rssend0,ar2

    lar     ar3,rsosize ; setup pointers and size
    mar     *,ar3
    lacl    rsosize
    add     #2
    mar     *-,ar2
    sacl    rsochksum
    send

    lacc    #rsbufout   ;reset buffer pointer
    add     #rsbufout
    sacl    rsobufpnt

rssend1lacl     rsobufpnt   ;divide 16-bit buffer to 8-bit access
    sfr                 ; output data
    sacl    rstemp2
    lar     ar2,rstemp2
    lacc    *,8
    sacl    *,ar3
    bsar    8
    bsar    8
    sacb
    send

    lacl    rsobufpnt   ; increment buffer pointer
    add     #1
    sacl    rsobufpnt
```

```
    lacb                ; update checksum
    add     rsochksum
    sacl    rsochksum

    banz    rssend1,ar2 ; loop until done

    xor     #000ffh     ; send checksum
    add     #1
    send

rssendeldp      #0          ; leave
    ret

; RSSTAT: get RS232 transmission, receiver status
rsstatldp       #rspage
    lacl    rspackok    ; check if packet OK
    bcnd    rsstato,neq

$1  lacl    rspackbad
    bcnd    $2,eq

    lacl    #2

rsstato                     ; if error, reset local variables
    sacb

    lacl    #0
    sacl    rscount
    sacl    rspackok
    sacl    rspackbad
    sacl    rschksum
    lacc    #rsbufin
    add     #rsbufin
    sacl    rsbufpnt

    lacb

$2
rsstate  ldp     #0         ; return 0 if all OK
    ret

; RSDUMP: dump variables and buffers for debugging
rsdumpldp       #rspage

    lar     ar2,#rssize
    lar     ar3,#29
    mar     *,ar2

rsdump0lacl     *+
    call    debugws

    mar     *,ar3
    banz    rsdump0,ar2

    call    debugr

    ldp     #0
    ret
```

```
;RS232TR: Table of reversed bits for MSB transmission
rs232tr .word    00080h,040c0h,020a0h,060e0h,01090h,050d0h,030b0h,070f0h
    .word    00888h,048c8h,028a8h,068e8h,01898h,058d8h,038b8h,078f8h
    .word    00484h,044c4h,024a4h,064e4h,01494h,054d4h,034b4h,074f4h
    .word    00c8ch,04ccch,02cach,06cech,01c9ch,05cdch,03cbch,07cfch
    .word    00282h,042c2h,022a2h,062e2h,01292h,052d2h,032b2h,072f2h
    .word    00a8ah,04acah,02aaah,06aeah,01a9ah,05adah,03abah,07afah
    .word    00686h,046c6h,026a6h,066e6h,01696h,056d6h,036b6h,076f6h
    .word    00e8eh,04eceh,02eaeh,06eeeh,01e9eh,05edeh,03ebeh,07efeh
    .word    00181h,041c1h,021a1h,061e1h,01191h,051d1h,031b1h,071f1h
    .word    00989h,049c9h,029a9h,069e9h,01999h,059d9h,039b9h,079f9h
    .word    00585h,045c5h,025a5h,065e5h,01595h,055d5h,035b5h,075f5h
    .word    00d8dh,04dcdh,02dadh,06dedh,01d9dh,05dddh,03dbdh,07dfdh
    .word    00383h,043c3h,023a3h,063e3h,01393h,053d3h,033b3h,073f3h
    .word    00b8bh,04bcbh,02babh,06bebh,01b9bh,05bdbh,03bbbh,07bfbh
    .word    00787h,047c7h,027a7h,067e7h,01797h,057d7h,037b7h,077f7h
    .word    00f8fh,04fcfh,02fafh,06fefh,01f9fh,05fdfh,03fbfh,07fffh

    .end
```

D.1.3.9 fpga.asm

```
; Use register assignments
    .mmregs

; Include global references
    .include "main.inc"
    .include "fpga.inc"
    .include "debug.inc"

; Location of FPGA bitstream in program memory
fpgaprg.set    08000h

; IO locations for out/in commands for static lines
fcclk.set    02001h
fdin.set    02002h
fprog.set    02003h

fdone.set    02000h
finit.set    02001h
fcdout.set    02002h

pae .set    02004h

; Variables
point.usect "fpga",1,0
temp.usect "fpga",1,0
size.usect "fpga",1,0
data.usect "fpga",1,0

; Setup FPGA data page
fpgapage.set   point

    .newblock

; FPGAINIT: place FPGA in reset mode
fpgainitldp     #fpgapage
```

```
    clrc    xf              ; set FPGA CCLK and PROG lines low
    out     temp,fcclk
    out     temp,fprog

    ldp     #0
    ret

    .newblock

; FPGALD: load fpga bitstream
fpgaldldp     #fpgapage

    lacl    #047h           ; Output "G" to terminal
    call    debuga

    clrc    xf              ; set FPGA CCLK and PROG lines low
    out     temp,fcclk
    out     temp,fprog

    rpt     #1000           ; wait for FPGA reset
    nop

    setc    xf              ; set PROG high
    out     temp,fprog

    rpt     #100            ; wait...
    nop

    lar     ar3,#0ffffh
    mar     *,ar3

$1  in      temp,finit      ; wait for INIT to be HIGH
    bcnd    $8,bio          ;  wait 65536 cycles
    b       $9              ;  if still low after 65536 cycles, FPGA probably not
$8  banz    $1              ;    in socket!  ERROR OUT!
    b       $7

$9  lacc    #fpgaprg        ; first byte of bitstream size
    sacl    point
    tblr    size

    add     #1              ; increment read pointer
    sacl    point

    lacl    size            ; output bitstream size to debugger
    call    debugwr

$2  lacl    point           ; get a word from memory
    tblr    data
    add     #1
    sacl    point           ; increment pointer

    lar     ar3,#15         ; set counter to 15 (16 bits to send)
    mar     *,ar3

$3  clrc    xf              ; set CCLK low
    out     temp,fcclk

    in      temp,finit      ; ERROR OUT if INIT LOW
```

```
    bcnd    $7,bio

    rpt     #20             ; wait
    nop

    bit     data,0          ; set DIN based on MSB of data
    bcnd    $4,ntc
    setc    xf

$4  out     temp,fdin

    setc    xf
    out     temp,fcclk

    rpt     #20             ; wait
    nop

    lacl    data            ; shift data left 1 bit
    sfl
    sacl    data

    sar     ar3,temp        ; on 8th bit, decrement size counter
    lacl    temp
    xor     #08h
    bcnd    $5,neq

    lacl    size
    sub     #1
    sacl    size
    bcnd    $6,eq

$5  mar     *,ar3           ; do the rest of the bits
    banz    $3

    lacl    size            ; decrement size counter
    sub     #1
    sacl    size
    bcnd    $6,eq           ; loop until complete stream is done

    b       $2

$6  lacl    #044h           ; Output a "D" when done programming
    call    debugar

    lacl    #0              ; leave with error code 0

    ldp     #0
    ret

$7  lacl    #045h           ; Output a "E" for error
    call    debugar

    ldp     #fpgapage

    lacl    size            ; Output stream location
    call    debugwr

    lacl    #1              ; leave with error code 1
```

```
    ldp     #0
    ret

    .newblock

; FPGAPUT: set 13 bit accumulator value to FPGA
fpgaput
    ldp     #fpgapage       ; Set FPGA data page

    setc    intm            ; disable interrupts

    sfl                     ; shift accumulator left 2 bits
    sfl
    and     #07ffch         ; set 3 other bits to 0 (bit 15 start bit)
    or      #00003h         ; set stop bits
    sacl    data            ; store it

    lar     ar3,#15         ; set counter for 16 bits
    mar     *,ar3

$3  clrc    xf              ; set CDCLK low
    out     temp,fcclk

    rpt     #5              ; wait...
    nop

    bit     data,0          ; set CDIN based on high bit of data
    bcnd    $4,ntc
    setc    xf

$4  out     temp,fdin

    setc    xf              ; set CDCLK high
    out     temp,fcclk

    rpt     #5              ; wait
    nop

    lacl    data            ; left shift data 1 bit
    sfl
    sacl    data

$5  mar     *,ar3           ; do all 16 bits
    banz    $3

    clrc    intm            ; enable interrupts

    ldp     #0              ; leave
    ret

        .newblock

;FPGAGET: Get an 8 bit value from the FPGA
fpgaget
    ldp     #fpgapage       ; set FPGA data page

    setc    intm            ; disable interrupts

    lar     ar3,#7          ; set for 8 bits
```

```
    mar     *,ar3

    setc    xf              ; set CDIN low
    out     temp,fdin

    lacl    #255            ; reset data variable
    sacl    data

$3  clrc    xf              ; set CDCLK low
    out     temp,fcclk

    lacl    data            ; shift data left 1 bit
    sfl
    sacl    data

    rpt     #5              ; wait...
    nop

    setc    xf              ; set CDCLK high
    out     temp,fcclk

    in      temp,fcdout     ; set bit 0 of data based on CDOUT

    bcnd    $4,bio
    opl     #00001h,data

$4
    rpt     #5              ; wait...
    nop

    mar     *,ar3           ; do the other 7 bits
    banz    $3

    lacl    data            ; move data received into accumulator

    clrc    intm            ; enable interrupts

    ldp     #0
    ret                     ; return

    .newblock

;FPGADUMP: output contents of FIFO to terminal
fpgadump
    ldp     #fpgapage       ; set FPGA data page

    in      temp,pae        ; check almost empty line
    bcnd    $2,bio          ;  leave if almost empty

    mar     *,ar3           ; set memory for data move
    lar     ar3,#infpga
    setc    intm            ; disable interrupts
    rpt     #4095           ; move 4096 words
    in      *+,04000h
    clrc    intm            ; enable interrupts

    lacl    #02eh           ; print a "." to the terminal
    call    debuga
```

```
    lar     ar3,#infpga    ; examine data
    lar     ar2,#2047
    mar     *,ar3

$1
; Uncomment to output FIFO data to terminal

;   lacl    *+
;   call    debugws
;   mar     *,ar3
;   lacl    *+
;   call    debugws

; Uncomment to check bit 27 (new line code)
;   mar     *+
;   bit     *+,5
;   bcnd    $3,ntc
;   lacl    #02dh
;   call    debuga

$3  mar     *,ar2
    banz    $1,ar3

$2
    ldp     #0             ; leave
    ret

    .newblock

;FPGAREC: Get 252 words (126 quadlets) from FIFO
fpgarec
    ldp     #fpgapage      ; set FPGA data page

    lacl    #1             ; set error code to 1

    in      temp,pae       ; leave is FIFO almost empty
    bcnd    $2,bio

    mar     *,ar3
    lar     ar3,#infpga
    setc    intm           ; disable interrupts
    rpt     #fpksizew-1     ; read 256 words in
    in      *+,04000h
    clrc    intm           ; enable interrupts

; trying something

    lar     ar3,#infpga    ; examine data
    lar     ar2,#fpksizeq-1
    mar     *,ar3

$1

    mar     *+
    bit     *+,5
    bcnd    $3,ntc
    mar *-
    bit     *+,6
    bcnd    $3,ntc
```

```
    lacl    #02dh
    call    debuga

$3  mar     *,ar2
    banz    $1,ar3

    lacl    #0              ; set error code to 0 (success)
$2
    ldp     #0              ; leave
    ret

    .end
```

D.1.3.10 tsb.asm

```
; Use register assignments
    .mmregs

; Include global references
    .include "tsb.inc"
    .include "main.inc"
    .include "debug.inc"

;ISO transmission header
isohead.set    0000h

; 1394 stack variables
grfar2.usect  "grfdata",1,0
grfar3.usect  "grfdata",1,0
grfar4.usect  "grfdata",1,0
grfar5.usect  "grfdata",1,0
grfrp.usect  "grfdata",1,0
grfwp.usect  "grfdata",1,0
grfintl.usect  "grfdata",1,0
grfinth.usect  "grfdata",1,0
grftsbl.usect  "grfdata",1,0
grftsbh.usect  "grfdata",1,0
grffirst.usect  "grfdata",1,0
grfsize.usect  "grfdata",1,0
grfoffst.usect  "grfdata",1,0
grfpnt.usect  "grfdata",1,0
grfpdwsr.usect  "grfdata",1,0
grfgreg.usect  "grfdata",1,0
grfvecl.usect  "grfdata",1,0
grfbrd.usect  "grfdata",1,0
grfetc.usect  "grfdata",1,0
grfstatl.usect  "grfdata",1,0
grfstath.usect  "grfdata",1,0
grfwdone.usect  "grfdata",1,0
grfwtlab.usect  "grfdata",1,0
grfwsrid.usect  "grfdata",1,0
grfwdatl.usect  "grfdata",1,0
grfwdath.usect  "grfdata",1,0
grfflag.usect  "grfdata",1,0
grfadd.usect  "grfdata",1,0
grfind.usect  "grfdata",1,0
grfdoh.usect  "grfdata",1,0
grfdid.usect  "grfdata",1,0
grftsize.usect  "grfdata",1,0
```

```
tsbtlab.usect  "grfdata",1,0
tsbl.usect  "grfdata",1,0
tsbh.usect  "grfdata",1,0
tsbwcntl.usect  "grfdata",1,0
tsbwcnth.usect  "grfdata",1,0
hostpntl.usect  "grfdata",1,0
hostpnth.usect  "grfdata",1,0
hostcntl.usect  "grfdata",1,0
hostcnth.usect  "grfdata",1,0
hostcmsk.usect  "grfdata",1,0

grfnodel.usect  "grfdata",1,0
grfnodeh.usect  "grfdata",1,0

isook.usect  "grfdata",1,0

tempc.usect  "grfdata",1,0

;sendloc.usect  "grfdata",1,0
;lastloc.usect  "grfdata",1,0
;sendsize.usect  "grfdata",1,0
;lastsize.usect  "grfdata",1,0
;sendlab.usect  "grfdata",1,0
;lastlab.usect  "grfdata",1,0
;sendvec.usect  "grfdata",1,0
;recount.usect  "grfdata",1,0

grffifo.usect  "grffifo",0100h,0
grfpack.usect  "grfpack",0400h,0
grftran.usect  "grftran",0400h,0

sendbuf1.usect  "sendbuf",0300h,0
sendbuf2.usect  "sendbuf",0300h,0

; Set GRF page
grfpage.set grfar2

rintto.set 01000h
wdoneto.set 08000h

; TSB12C01A registers
rversion.set 06000h
rnode.set 06002h
rcontrol.set 06004h
rinter.set 06006h
rintmask.set 06008h
rphyacc.set 06012h
ratfstat.set 06018h
ritfstat.set 0601ah
rgrfstat.set 0601eh
ratf1.set 06040h
ratf2.set 06042h
ratf3.set 06046h
ritf1.set 06048h
ritf2.set 0604ah
ritf3.set 0604eh
rgrf.set 06060h

; Register initializers
```

```
dcontrol1.set 00201h
dcontrolh.set 0c630h
dinterl.set 00600h
dinterh.set 00308h
dintmaskl.set 00600h
dintmaskh.set 08308h
datfstatl.set 000c8h
dgrfstatl.set 00064h
ditfstatl.set 000c8h
datfstat2.set 010c8h
dgrfstat2.set 01064h
ditfstat2.set 010c8h

    .text

    .newblock

;TSBINIT: setup interrupt services for the TSB12C01A chip
tsbinit
    ldp     #0
    setc    intm            ; disable interrupts

    lacc    #isr            ; set interrupt handler
    sacl    060h

    ldp     #grfpage

    clrc    xf
    out     grftsbl,02000h  ; hold down reset
    rpt     #65535          ; wait...
    nop

    setc    xf
    out     grftsbl,02000h  ;release reset

;   mar     *,ar2
;   lar     ar2,#0382
$1
    rpt     #65535          ; wait...
    nop
;   banz    $1

    lacl    #01h            ; set interrupt flags
    samm    imr
    samm    ifr

    in      grftsbl,rversion   ; try to read something from the 1394 LLC
    in      grftsbh,rversion+1

    lacl    #031h             ; debugging stuff
    call    debuga
    ldp     #grfpage

    splk    #dgrfstatl,grftsbl  ; Set GRF fifo size
    splk    #00000h,grftsbh
    out     grftsbl,rgrfstat
    out     grftsbh,rgrfstat+1
    splk    #datfstatl,grftsbl  ; Set ATF fifo size
    out     grftsbl,ratfstat
```

```
   out     grftsbh,ratfstat+1
   splk    #ditfstat1,grftsbl  ; Set ITF fifo size
   out     grftsbl,ritfstat
   out     grftsbh,ritfstat+1

   lacl    #033h
   call    debuga
   ldp     #grfpage

   splk    #dgrfstat2,grftsbl  ; Set GRF fifo size and clear
   splk    #00000h,grftsbh
   out     grftsbl,rgrfstat
   out     grftsbh,rgrfstat+1
   splk    #datfstat2,grftsbl  ; Set ATF fifo size and clear
   out     grftsbl,ratfstat
   out     grftsbh,ratfstat+1
   splk    #ditfstat2,grftsbl  ; Set ITF fifo size and clear
   out     grftsbl,ritfstat
   out     grftsbh,ritfstat+1

   lacl    #035h
   call    debuga
   ldp     #grfpage

   splk    #dinterl,grftsbl    ; Set interrupt registers
   splk    #dinterh,grftsbh
   out     grftsbl,rinter
   out     grftsbh,rinter+1
   out     grftsbl,rintmask
   out     grftsbh,rintmask+1

   lacl    #036h
   call    debuga
   ldp     #grfpage

   splk     #dcontroll,grftsbl ; Set control registers
   splk     #dcontrolh,grftsbh
   out     grftsbl,rcontrol
   out     grftsbh,rcontrol+1

   lacl    #037h
   call    debuga
   ldp     #grfpage

   lacc    #grffifo            ; initialize variables
   sacl    grfrp
   sacl    grfwp
   lacc    #grfpack
   sacl    grfpnt
   lacl    #0
   sacl    grfsize
   sacl    grfflag
   lacl    #1
   sacl    grffirst
   sacl    grfwdone
   lacl    #3
   sacl    grfoffst
;  lacc    #0fc00h            ; async trans supprt of large data removed
;  sacl    tsbtlab
```

```
;   lacc    #sendbuf1
;   sacl    sendloc
;   lacl    #05
;   sacl    recount
    lacl    #0
    sacl    isook

    splk    #00000h,grftsbl     ; Send bus reset
    splk    #0417fh,grftsbh
    out     grftsbl,rphyacc
    out     grftsbh,rphyacc+1

$2
    in      grftsbl,rphyacc     ; Wait for phy write
    in      grftsbh,rphyacc+1
    lacl    grftsbh
    and     #04000h
    bcnd    $2,neq

;   mar     *,ar2
;   lar     ar2,#0100
$3
    rpt     #65535              ; Wait...
    nop
;   banz    $3

    splk    #dintmaskl,grftsbl  ; set interrupt registers
    splk    #dintmaskh,grftsbh
    out     grftsbl,rintmask
    out     grftsbh,rintmask+1

    ldp     #0

    clrc    intm            ; enable interrupts

    ret                     ; return

    .newblock

;ISR: Interrupt Service Routine for LLC (Receives and Asycn acknowledgments)
isr
    ldp     #grfpage        ; set 1394 data page

    lamm    pdwsr           ; consider debugger settings
    sacl    grfpdwsr
    lamm    greg
    sacl    grfgreg

    lacl    #0
    samm    pdwsr
    samm    greg

    sar     ar2,grfar2      ; save ar's used
    sar     ar3,grfar3
    sar     ar4,grfar4
    sar     ar5,grfar5

;   lacl    grfflag         ; debugging code
```

```
;   add     #1
;   sacl    grfflag

isrloop
    in      grftsbl,02006h   ; is anything available on the GRF
    bcnd    isr0,bio

    in      grfintl,rinter   ; if not, check other possible interrupts
    in      grfinth,rinter+1

    bit     grfinth,12       ; is async trans stuck?
    bcnd    isratsk,tc

    bit     grfintl,5
    bcnd    isrigo,tc        ; iso cycle started

    bit     grfintl,6
    bcnd    isristop,tc      ; iso cycle ended

;   bit     grfintl,8        ; iso cycle lost
;   bcnd    isrislst,tc

;   bit     grfinth,6        ; data received
;   bcnd    isr0,tc

    b       isrleave         ; nothing else, leave

isr0
    in      grfstatl,rgrfstat   ;get quadlet status
    in      grfstath,rgrfstat+1
    bit     grfstath,15
    bcnd    isrleave,tc         ; if empty, leave

;   in      grftsbl,02006h      ; check direct line optionally also
;   bcnd    isr1,bio
;   b       isrleave

isr1
    in      grftsbl,rgrf        ;get quadlet
    in      grftsbh,rgrf+1

    mar     *,ar2

    lar     ar2,grfpnt          ; get buffer pointers
    lar     ar3,grfpnt

    lacl    grftsbl             ; store in packet buffer
    sacl    *+
    lacl    grftsbh
    sacl    *+
    sar     ar2,grfpnt          ; update pointer

    mar     *,ar3

    lacl    grffirst            ; is this the first quadlet of a packet?
    bcnd    isr2,eq

    bit     grfstatl,0          ; do not process as first unless "cd" bit set
    bcnd    isr6,tc
```

```
        b       isrnew

isr6
    lacl    #0                  ; unset first flag
    sacl    grffirst

    lacl    *                   ; determine tcode, lookup pre process code address
    and     #000f0h
    rpt     #3
    sfr
    sacl    grfoffst
    add     #grftbpre
    tblr    grftsbl
    lacl    grftsbl
    bacc                        ; call pre process

isr2
    lacl    grfsize             ; decrement size count (set by pre process call)
    sub     #1
    sacl    grfsize
    bcnd    isr3,eq             ; if packet complete, it may be debuggable

    lacl    grfoffst            ; lookup during process
    add     #grftbdur
    tblr    grftsbl
    lacl    grftsbl
    bacc                        ; call during process

isr3
    lacl    grfoffst            ; lookup tcode in debug table
    add     #grftbrec
    tblr    grftsbl
    lacl    grftsbl
    bcnd    isr4,eq             ; if set, record packet into debug fifo

    lar     ar4,#grfpack

isr5
    lar     ar5,grfwp           ; setup write to debug fifo
    mar     *,ar4
    lacl    *+,ar5
    sacl    *
    sar     ar4,grftsbh

    lacl    grfwp               ; adjust pointer for wrapping
    add     #1
    and     #000ffh
    or      #grffifo
    sacl    grfwp

    lacl    grftsbh
    xor     grfpnt
    bcnd    isr5,neq

    mar     *,ar3

isr4
    lacl    grfoffst              ; lookup tcode in post process table
```

```
    add     #grftbpos
    tblr    grftsbl
    lacl    grftsbl
    bacc                            ; call post process

isratsk
    splk    #010c8h,grftsbl         ; Set ATF fifos size and clear
    splk    #00000h,grftsbh
    out     grftsbl,ratfstat
    out     grftsbh,ratfstat+1

isrnew
    lacl    #1                      ; initialize some variables
    sacl    grffirst
    lacl    #3
    sacl    grfoffst
    lacc    #grfpack
    sacl    grfpnt

isrnext
    in      grftsbl,02006h          ; check for more on GRF
    bcnd    isr1,bio

isrleave
    splk    #0ffffh,grftsbl     ; set interrupt registers as begin handles
    out     grftsbl,rinter
    out     grftsbl,rinter+1

    lar     ar5,grfar5          ; restore ar's
    lar     ar4,grfar4
    lar     ar3,grfar3
    lar     ar2,grfar2

    lacl    grfgreg             ; restore environment for debugger
    samm    greg
    lacl    grfpdwsr
    samm    pdwsr

    lacl    #01h                ; set interrupt flag
    samm    ifr

    rete                        ; leave

isrigo
    lacl    #1                  ; set ISO flag ok to send
    sacl    isook

    splk    #00400h,grftsbl     ; set LLC interrupt registers
    splk    #00000h,grftsbh
    out     grftsbl,rinter
    out     grftsbh,rinter+1

    b       isrloop

isristop
    lacl    #0          ; set ISO flag no good to send
    sacl    isook

    splk    #00200h,grftsbl     ; set LLC interrupt registers
```

```
    splk    #00000h,grftsbh
    out     grftsbl,rinter
    out     grftsbh,rinter+1

    b       isrloop

isrislst
    splk    #dcontroll,grftsbl ; Set control registers
    splk    #dcontrolh,grftsbh
    out     grftsbl,rcontrol
    out     grftsbh,rcontrol+1

    splk    #00080h,grftsbl    ; set LLC interrupt registers
    splk    #00000h,grftsbh
    out     grftsbl,rinter
    out     grftsbh,rinter+1

    lacl    #1                 ; set debugger flag
    sacl    grfflag

    b       isrloop

; grftbrec: set to record received data into debugger fifo
grftbrec
    .word   0,0,0,0
    .word   0,0,0,0
    .word   0,0,0,0
    .word   0,0,0,0

; grftbpre: pre-calls
grftbpre
    .word   tc0_pr ,tc1_pr ,tc2_pr ,isrnext
    .word   tc4_pr ,tc5_pr ,tc6_pr ,isrnext
    .word   isrnext,tc9_pr ,isrnext,isrnext
    .word   isrnext,isrnext,tce_pr ,isrnext

; grftbdur: during-calls
grftbdur
    .word   isrnext,tc1_dr ,isrnext,isrnext
    .word   isrnext,isrnext,isrnext,isrnext
    .word   isrnext,tc9_dr ,isrnext,isrnext
    .word   isrnext,isrnext,tce_dr ,isrnext

; grftbpos: post-calls
grftbpos
    .word   tc0_po ,tc1_po ,tc2_po ,isrnext
    .word   tc4_po ,tc5_po ,tc6_po ,isrnext
    .word   isrnext,tc9_po ,isrnext,isrnext
    .word   isrnext,isrnext,isrnext,isrnext

; Banks: FFFF FFFn xxxx
; n=  0 CSR area
;     1 DSP Data Low  ($0000-$7FFF)
;     2 DSP Data High ($8000-$FFFF)
;     3 DSP Prog Low  ($0000-$7FFF)
;     4 DSP Prog High ($8000-$FFFF)
; n=  others, results in bad_address

; tcode 0 and 6 pre call
```

```
tc6_pr
tc0_pr
    lacl    #04h                ; used by 0 and 6
    sacl    grfsize             ; set grfsize to 4 quadlets
    b       isrnext

; tcode 0 post call (write quadlet)
tc0_po
    lar     ar3,#grfpack
    lar     ar2,#grftran
    lacl    *+,ar2              ; get tlabel,rt,tcode,priority
    and     #0fc00h
    or      #00020h             ; set to write response
    sacl    *+,ar3
    mar     *+,ar2              ; skip destination id
    lacl    #00h                ; set speed to 100Mb
    sacl    *+,ar3
    lacl    *+,ar2              ; get destoffsethigh
    sacl    grfdoh
    lacl    #0                  ; set r_code to resp_complete for now
    sacl    *+,ar3
    lacl    *+,ar2              ; get source id, check for broadcast
    sacl    *+

    and     #0003fh
    xor     #0003fh
    sacl    grfbrd              ; set broadcast flag, 0 if broadcast

    lacl    #0
    sacl    grftsize            ; set number of quads to send -3

    lacl    #0                  ; fill dol with nulls
    sacl    *+
    sacl    *+,ar3

    lacl    *+                  ; check lowword of lowoffset
    and     #0fffch             ; set to modulo 4
    sfr                         ; scale down
    sacl    grfadd

    lacl    grfdoh              ; Check highoffset = ffff
    xor     #0ffffh
    bcnd    tcw_ba,neq

    lacl    *                   ; Check highword of lowoffset = f00x (x = bit
0,1,2)
    and     #0fff8h
    xor     #0f000h
    bcnd    tcw_ba,neq

    lacl    *+
    and     #00007h

    add     #tc0_tb             ; determine what to do on which bank
    tblr    grfvecl
    lacl    grfvecl
    bacc

tc0_tb
```

```
    .word   tcw_ba ,tc0_fn1,tc0_fn2,tc0_fn3
    .word   tc0_fn4,tcw_ba ,tcw_ba ,tcw_ba

tc0_fn2
    opl     #08000h,grfadd

tc0_fn1
    lar     ar5,grfadd

    lacl    *+,ar5
    sacl    *+,ar3
    lacl    *+,ar5
    sacl    *+

    b       tc0_end

tc0_fn4
    opl     #08000h,grfadd

tc0_fn3
    lacl    grfadd
    rpt     #1
    tblw    *+

tc0_end

    lacl    grfbrd
    bcnd    isrnew,eq

    b       itransmit

tcw_ba
    lacc    #07000h             ; error in address space r_code=resp_address_error

tcw_ba0
    sacb

    lacl    grfbrd
    bcnd    isrnew,eq

    lar     ar2,#grftran+2
    lacb
    sacl    *

    b       itransmit

; tcode 1 and 9 pre call
tc9_pr
tc1_pr
    lacl    #0ffh               ; grfsize=255, will use during call
tc1_pr0
    sacl    grfsize
    b       isrnext

; tcode 1 and 9 during call
tc9_dr
tc1_dr
    lacl    grfsize
    xor     #000fch
```

```
    bcnd    isrnext,neq
    lacl    *+
    sacl    grfetc              ; store extended_tcode

    lacl    *                   ; get datalength
    sfr
    sfr
    add     #2                  ; compensate for ack quadlet too
    sacl    grfsize
    lacl    *
    and     #03h
    bcnd    tc1_dr0,neq
    lacl    grfsize             ; calculate number of quadlets left to read
    sub     #1
    sacl    grfsize
tc1_dr0
    b       isrnext

; tcode 1 post call (write block)
tc1_po
    lar     ar3,#grfpack
    lar     ar2,#grftran
    lacl    *+,ar2              ; get tlabel,rt,tcode,priority
    and     #0fc00h
    or      #00020h             ; set to write response
    sacl    *+,ar3
    mar     *+,ar2              ; skip destination id
    lacl    #00h                ; set speed to 100Mb
    sacl    *+,ar3
    lacl    *+,ar2              ; get destoffsethigh
    sacl    grfdoh
    lacl    #0                  ; set r_code to resp_complete for now
    sacl    *+,ar3
    lacl    *+,ar2              ; get source id, check for broadcast
    sacl    *+

    and     #0003fh
    xor     #0003fh
    sacl    grfbrd              ; set broadcast flag, 0 if broadcast

    lacl    #0
    sacl    grftsize            ; set number of quads to send -3

    lacl    #0                  ; fill dol with nulls
    sacl    *+
    sacl    *+,ar3

    lacl    *+                  ; check lowword of lowoffset
    and     #0fffch             ; set to modulo 4
    sfr                         ; scale down
    sacl    grfadd

    lacl    grfdoh              ; Check highoffset = ffff
    xor     #0ffffh
    bcnd    tcw_ba,neq

    lacl    *                   ; Check highword of lowoffset = f00x (x = bit 0,1,2)
    and     #0fff8h
    xor     #0f000h
```

```
    bcnd    tcw_ba,neq

    lacl    *+
    and     #00007h
    sacl    grfind

    mar     *+                  ; skip extended_tcode

    lacl    *+                  ; get datalength
    sacl    grfsize
    bcnd    tcw_ba0,eq          ; handle datalength=0
    add     #3
    sfr
    sfr
    sub     #1
    sacl    grfdoh              ; set size for repeat

    lacl    grfind              ; determine what to do on which bank
    add     #tc1_tb
    tblr    grfvec1
    lacl    grfvec1
    lar     ar4,grfdoh
    bacc

tc1_tb
    .word   tcw_ba ,tc1_fn1,tc1_fn2,tc1_fn3
    .word   tc1_fn4,tcw_ba ,tcw_ba ,tcw_ba

tc1_fn2
    opl     #08000h,grfadd

tc1_fn1

    lar     ar5,grfadd

tc1_fn1a
    lacl    *+,ar5
    sacl    *+,ar3
    lacl    *+,ar5
    sacl    *+,ar4

    banz    tc1_fn1a,ar3

    b       tc0_end

tc1_fn4
    opl     #08000h,grfadd

tc1_fn3
    lacl    grfadd

tc1_fn3a
    rpt     #1
    tblw    *+

    add     #2
    mar     *,ar4

    banz    tc1_fn3a,ar3
```

```
    b       tc0_end

; tcode 2 and 6 post call
tc6_po
tc2_po
    lar     ar3,#grfpack
    lacl    *+                  ; get tlabel
    and     #0fc00h
    xor     grfwtlab
    bcnd    isrnew,neq          ; check tlabel
    adrk    #2
    lacl    *+                  ; get source id
    xor     grfwsrid
    bcnd    isrnew,neq          ; check source id

    adrk    #2                  ; move received quad into memory, not nec for write
response
    lacl    *+
    sacl    grfwdatl
    lacl    *
    sacl    grfwdath

    lacl    #01h
    sacl    grfwdone            ; indicate transaction completed
;   sacl    grfflag

    b       isrnew

; tcode 2 and 4 pre call
tc2_pr
tc4_pr
    lacl    #03h                ; used by 2 and 4
    sacl    grfsize
    b       isrnext

; tcode 4 post call (read quadlet)
tc4_po
    lar     ar3,#grfpack
    lar     ar2,#grftran
    lacl    *+,ar2              ; get tlabel,rt,tcode,priority
    and     #0fc00h
    or      #00060h             ; set to read response
    sacl    *+,ar3
    mar     *+,ar2              ; skip destination id
    lacl    #00h                ; set speed to 100Mb
    sacl    *+,ar3
    lacl    *+,ar2              ; get destoffsethigh
    sacl    grfdoh
    lacl    #0                  ; set r_code to resp_complete for now
    sacl    *+,ar3
    lacl    *+,ar2              ; get source id, check for broadcast
    sacl    *+

    and     #0003fh
    xor     #0003fh
    bcnd    isrnew,eq           ; if this is a broadcast, leave now

    lacl    #1
```

```
    sacl    grftsize            ; set number of quads to send -3

    lacl    #0                  ; fill dol with nulls
    sacl    *+
    sacl    *+,ar3

    lacl    *+                  ; check lowword of lowoffset
    and     #0fffch             ; set to modulo 4
    sfr                         ; scale down
    sacl    grfadd

    lacl    grfdoh              ; Check highoffset = ffff
    xor     #0ffffh
    bcnd    tcr_ba,neq

    lacl    *                   ; Check highword of lowoffset = f00x (x = bit 0,1,2)
    and     #0fff8h
    xor     #0f000h
    bcnd    tcr_ba,neq

    lacl    *+
    and     #00007h

    add     #grftc4tb           ; determine what to do on which bank
    tblr    grfvec1
    lacl    grfvec1
    mar     *,ar5
    bacc

grftc4tb
    .word   tc4_fn0,tc4_fn1,tc4_fn2,tc4_fn3
    .word   tc4_fn4,tcr_ba ,tcr_ba ,tcr_ba

tc4_fn2
    opl     #08000h,grfadd

tc4_fn1
    lar     ar5,grfadd

    lacl    *+,ar2
    sacl    *+,ar5
    lacl    *+,ar2
    sacl    *+,ar5

    b       itransmit

tc4_fn4
    opl     #08000h,grfadd

tc4_fn3
    mar     *,ar2

    lacl    grfadd
    tblr    *+
    add     #1
    tblr    *

    b       itransmit
```

```
tc4_fn0
    mar     *,ar2

    lacl    grfadd              ; error if msb=0 and offset<400
    sub     #0200h
    sacl    grfadd
    bcnd    tcr_ba,nc

    sub     #0200h              ; error if msb=0 and offset>800
    bcnd    tcr_ba,c

    lacl    grfadd
    add     #grfcsrom+1
    tblr    *+
    sub     #1
    tblr    *

    b       itransmit

tcr_ba
    lacc    #07000h             ; error in address space r_code=resp_address_error

tcr_ba0
    sacb

    lacl    #1
    sacl    grftsize            ; set number of quads to send -3

    lacl    #0
    mar     *,ar2

    sacl    *+
    sacl    *+

    lar     ar2,#grftran+2
    lacb
    sacl    *

    b       itransmit

; tcode 5 pre call
tc5_pr
    lacl    #04h
    sacl    grfsize
    b       isrnext

; tcode 5 post call (read block)
tc5_po
    lar     ar3,#grfpack
    lar     ar2,#grftran
    lacl    *+,ar2              ; get tlabel,rt,tcode,priority
    and     #0fc00h
    or      #00070h             ; set to read block response
    sacl    *+,ar3
    mar     *+,ar2              ; skip destination id
    lacl    #00h                ; set speed to 100Mb
    sacl    *+,ar3
    lacl    *+,ar2              ; get destoffsethigh
    sacl    grfdoh
```

```
    lacl    #0                  ; set r_code to resp_complete for now
    sacl    *+,ar3
    lacl    *+,ar2              ; get source id, check for broadcast
    sacl    *+

    and     #0003fh
    xor     #0003fh
    bcnd    isrnew,eq           ; if this is a broadcast, leave now

    lacl    #0                  ; fill dol with nulls
    sacl    *+
    sacl    *+,ar3

    lacl    *+                  ; check lowword of lowoffset
    and     #0fffch             ; set to modulo 4
    sfr                         ; scale down
    sacl    grfadd

    lacl    grfdoh              ; Check highoffset = ffff
    xor     #0ffffh
    bcnd    tcr_ba,neq

    lacl    *                   ; Check highword of lowoffset = f00x (x = bit 0,1,2)
    and     #0fff8h
    xor     #0f000h
    bcnd    tcr_ba,neq

    lacl    *+
    and     #00007h
    sacl    grfind

    mar     *+                  ; skip extended_tcode

    lacl    *,ar2               ; get datalength
    sacl    grfsize
    bcnd    tcr_ba0,eq          ; handle datalength=0
    add     #3
    sfr
    sfr
    sub     #1
    sacl    grfdoh              ; set size for repeat
    add     #2
    sacl    grftsize            ; set transmission size

    lacl    grfind              ; determine what to do on which bank
    add     #grftc5tb
    tblr    grfvec1
    lacl    grfvec1
    lar     ar4,grfdoh
    bacc

grftc5tb
    .word   tc5_fn0,tc5_fn1,tc5_fn2,tc5_fn3
    .word   tc5_fn4,tcr_ba ,tcr_ba ,tcr_ba

tc5_fn2
    opl     #08000h,grfadd

tc5_fn1
```

```
    lacl    #0
    sacl    *+
    lar     ar5,grfadd
    lacl    grfsize
    sacl    *+,ar5

tc5_fn1a
    lacl    *+,ar2
    sacl    *+,ar5
    lacl    *+,ar2
    sacl    *+,ar4

        banz    tc5_fn1a,ar5

    b       itransmit

tc5_fn4
    opl     #08000h,grfadd

tc5_fn3
    lacl    #0
    sacl    *+
    lacl    grfsize
    sacl    *+
    lacl    grfadd

tc5_fn3a
    rpt     #1
    tblr    *+

    add     #2
    mar     *,ar4

    banz    tc5_fn3a,ar2

    b       itransmit

tc5_fn0
    lacl    grfadd              ; error if msb=0 and offset<400
    sub     #0200h
    sacl    grfadd
    bcnd    tcr_ba,nc

    sub     #0200h              ; error if msb=0 and offset>800
    bcnd    tcr_ba,c

    lacl    #0
    sacl    *+
    lacl    grfsize
    sacl    *+

    lacl    grfadd
    add     #grfcsrom+1

tc5_fn0a
    tblr    *+
    sub     #1
    tblr    *+,ar4
    add     #3
```

```
    banz    tc5_fn0a,ar2

    b       itransmit

; tcode 9 post call (lock)
tc9_po
    lar     ar3,#grfpack
    lar     ar2,#grftran
    lacl    *+,ar2              ; get tlabel,rt,tcode,priority
    and     #0fc00h
    or      #000b0h             ; set to lock response
    sacl    *+,ar3
    mar     *+,ar2              ; skip destination id
    lacl    #00h                ; set speed to 100Mb
    sacl    *+,ar3
    lacl    *+,ar2              ; get destoffsethigh
    sacl    grfdoh
    lacl    #0                  ; set r_code to resp_complete for now
    sacl    *+,ar3
    lacl    *+,ar2              ; get source id, check for broadcast
    sacl    *+

    and     #0003fh
    xor     #0003fh
    bcnd    isrnew,eq           ; if this is a broadcast, leave now

    lacl    #2
    sacl    grftsize            ; set number of quads to send -3

    lacl    #0                  ; fill dol with nulls
    sacl    *+
    sacl    *+,ar3

    lacl    *+                  ; check lowword of lowoffset
    and     #0fffch             ; set to modulo 4
    sfr                         ; scale down
    sacl    grfadd

    lacl    grfdoh              ; Check highoffset = ffff
    xor     #0ffffh
    bcnd    tc1_ba,neq

    lacl    *                   ; Check highword of lowoffset = f00x (x = bit 0,1,2)
    and     #0fff8h
    xor     #0f000h
    bcnd    tc1_ba,neq

    lacl    *+
    and     #00007h
    sacl    grfind

    lacl    *                   ; check extended_tcode
    sub     #8
    bcnd    tc1_ba,c            ; leave if e_tcode > 8

    lacl    *+                  ; check datalength vs e_tcode
    add     #tc9_tb0
    tblr    grfvec1
```

```
    lacl    *+,ar2              ; if datalength and table do not match, leave
    xor     grfvec1             ;  advance to block data
    bcnd    tc1_ba,neq

    lacl    grfind              ; determine what to do on which bank (read)
    add     #tc9_tb1
    tblr    grfvec1
    lacl    grfvec1
    bacc

; fetch_add changed to 8h since aic5800 stops NT with asynclocks of one quadlet
tc9_tb0
    .word   0ffffh,00008h,00008h,00008h,0ffffh,00008h,00008h,0ffffh ; no
little_add
    .word   0ffffh,0ffffh,0ffffh,0ffffh,0ffffh,0ffffh,0ffffh,0ffffh

tc9_tb1
    .word   tc1_ba ,tc9_fr1,tc9_fr2,tc9_fr3
    .word   tc9_fr4,tc1_ba ,tc1_ba ,tc1_ba

tc9_fr2
    opl     #08000h,grfadd

tc9_fr1
    lar     ar5,grfadd
    lacl    grfetc
    sacl    *+
    lacl    #04h
    sacl    *+,ar5

    lacl    *+,ar2
    sacl    *+,ar5
    sacl    grftsbl
    lacl    *,ar2
    sacl    *+,ar5
    sacl    grftsbh
    mar     *-

    b       tc9_po0

tc9_fr4
    opl     #08000h,grfadd

tc9_fr3
    lacl    grfetc
    sacl    *+
    lacl    #04h
    sacl    *+

    lacl    grfadd
    tblr    grftsbl
    add     #1
    tblr    grftsbh

    lacl    grftsbl
    sacl    *+
    lacl    grftsbh
    sacl    *+
```

```
;    b       tc9_po0

tc9_po0
    lacl    grfetc                  ;perform appropriate e_tcode function
    add     #tc9_tb2
    tblr    grfvec1
    lacl    grfvec1
    mar     *,ar3
    bacc

tc9_tb2
    .word   itransmit,tc9_ma,tc9_co,tc9_fe,itransmit,tc9_bo,tc9_wr,itransmit

tc9_ma
    lacl    *+                  ; get arg_value low
    xor     #0ffffh             ; compliment
    and     grftsbl             ; and with old_value low
    mar     *+
    or      *-                  ; or with data_value low
    sacl    grftsbl             ; store over old_value low
    lacl    *+                  ; get arg_value high
    xor     #0ffffh             ; compliment
    and     grftsbh             ; and with old_value high
    mar     *+
    or      *                   ; or with data_value high
    sacl    grftsbh             ; store over old_value high

    b       tc9_po1

tc9_co
    lacl    grftsbl             ; get old_value low
    xor     *+                  ; xor with arg_value low
    bcnd    itransmit,neq       ; if not zero, then not equal, leave
    lacl    grftsbh             ; get old_value high
    xor     *+                  ; xor with arg_value high
    bcnd    itransmit,neq       ; if not zero, then not equal, leave

tc9_co0
    lacl    *+                 ; move data_value into old_value
    sacl    grftsbl
    lacl    *
    sacl    grftsbh

    b       tc9_po1

tc9_bo
    lacl    grftsbl            ; get old_value low
    xor     *+                 ; xor with arg_value low
    bcnd    tc9_bo0,neq        ; if not zero, then not equal, advance to fetch_add
    lacl    grftsbh            ; get old_value high
    xor     *+                 ; xor with arg_value high
    bcnd    tc9_fe,neq         ; if not zero, then not equal, advance to fetch_add
    b       itransmit          ; leave

tc9_bo0
    mar     *+

tc9_fe
    lacc    grftsbh,16        ; load high acc with old_value high
```

```
    or      grftsbl          ; or low acc with old_value low
    add     *+               ; add arg_value low to low acc
    add     *,16             ; add arg_value high to high acc
    sacl    grftsbl          ; store acc back into old_value
    sach    grftsbh

    b       tc9_po1

tc9_wr
    lacl    grftsbl          ; get old_value low
    xor     *+               ; xor with arg_value low
    bcnd    tc9_bo0,neq      ; if not zero, then not equal, advance to fetch_add
    lacl    grftsbh          ; get old_value high
    xor     *+               ; xor with arg_value high
    bcnd    tc9_fe,neq       ; if not zero, then not equal, adcance to fetch_add
    b       tc9_co0          ; move data_value to old_value

tc9_po1
    lacl    grfind           ; determine what to do on which bank (read)
    add     #tc9_tb3
    tblr    grfvec1
    lacl    grfvec1
    mar     *,ar5
    bacc

tc9_tb3
    .word   isrnew,tc9_fw1,tc9_fw1,tc9_fw3
    .word   tc9_fw3,isrnew,isrnew,isrnew

tc9_fw1
    lacl    grftsbl          ; write back data memory
    sacl    *+
    lacl    grftsbh
    sacl    *

    b       itransmit

tc9_fw3
    lacl    grfadd           ; write back program memory
    tblw    grftsbl
    add     #1
    tblw    grftsbh

    b       itransmit

; send one quadlet payload since TSB12C01A or AIC5800 has a problem with no
payload

tc1_ba
    lacc    #07000h          ; error in address space r_code=resp_address_error

tc1_ba0
    sacb

    lacl    grfetc
    mar     *,ar2
    sacl    *+
    lacl    #04h
    sacl    *+
```

```
    lacl    #0
    sacl    *+
    sacl    *+

    lar     ar2,#grftran+2
    lacb
    sacl    *

    b       itransmit

; tcode 14 pre call
tce_pr
    lacl    #0ffh
    sacl    grfsize
    b       isrnext

; tcode 14 during call
tce_dr
    lacl    *+
    xor     #00001h
    bcnd    tce_pr,neq
    lacl    *+
    bcnd    tce_pr,neq

    lacc    #00000h
    sacl    grftsbl
    lacc    #08000h
    sacl    grftsbh
    out     grftsbl,rphyacc
    out     grftsbh,rphyacc+1

tce_dr0
    in      grftsbl,rphyacc   ; Wait for phy read
    in      grftsbh,rphyacc+1
    lacl    grftsbh
    and     #08000h
    bcnd    tce_dr0,neq

    lar     ar2,#0ffffh
    mar     *,ar2

tce_dr1
    banz    tce_dr1,ar2

    in      grftsbl,rphyacc
    in      grftsbh,rphyacc+1

    lacl    grftsbl
    sfr
    sfr
    or      #0ffc0h
    sacl    grfnodeh
    lacl    #00
    sacl    grfnodel

    out     grfnodel,rnode
    out     grfnodeh,rnode+1
```

```
    lacc    #00000h
    sacl    grftsbl
    lacc    #08100h
    sacl    grftsbh
    out     grftsbl,rphyacc
    out     grftsbh,rphyacc+1

tce_dr2
    in      grftsbl,rphyacc   ; Wait for phy read
    in      grftsbh,rphyacc+1
    lacl    grftsbh
    and     #08000h
    bcnd    tce_dr2,neq

;         in      grfflag,0x6012
;         in      grftsbh,0x6013

;         lacl    #046h
;         sacl    grftsbh
;         lacl    #00eh
;         sacl    grftsbl
;         out     grftsbl,ratf1
;         out     grftsbh,ratf1+1

;         lacl    grftsbh
;         xor     #0ffffh
;         sacl    grftsbh
;         lacc    #0ffffh
;         sacl    grftsbl
;         out     grftsbl,ratf3
;         out     grftsbh,ratf3+1

    b       isrnew

; itransmit: transmit an async packet when in isr routine
itransmit
    splk    #00000h,grftsbl
    out     grftsbl,rinter
    splk    #00400h,grftsbh
    out     grftsbh,rinter+1

    lar     ar2,#grftran
    mar     *,ar2
    lar     ar3,grftsize

    out     *+,ratf1
    out     *+,ratf1+1

itransmit0

    out     *+,ratf2
    out     *+,ratf2+1,ar3

    banz    itransmit0,ar2

    out     *+,ratf3
    out     *+,ratf3+1

itransmit2
```

```
    in      grfintl,rinter
    in      grfinth,rinter+1

    bit     grfinth,5
    bcnd    itransmit2,ntc

    in      grftsbl,rmode
    in      grftsbh,rmode+1

    lacl    grftsbl
    bsar    4
    and     #0fh
    bcnd    itransmit3,eq
    sub     #3
    bcnd    itransmit3,geq

    b       isrnew

itransmit3
;   lacl    grfflag
;   add     #1
;   sacl    grfflag

    b       itransmit


grfcsrom
    .word   00404h,0ffffh     ;Bus_Info_Block 0404
    .word   03133h,03934h
    .word   000ffh,01000h
    .word   00000h,0d101h     ;id 80ffff
    .word   00000h,00001h

    .word   00004h,0ffffh     ;Root dir
    .word   00380h,0ffffh      ;module_vendor_id
    .word   00c00h,00380h      ;node_capabilities 83c0
    .word   08d00h,00002h      ;node_unique_id
    .word   0d100h,00004h      ;unit_directory

    .word   00002h,0ffffh     ;Leaf 1, node_vendor_id
    .word   00000h,0d101h      ;id 80ffff
    .word   00000h,00001h

    .word   00003h,0ffffh     ;unit_directory
    .word   01200h,0a02dh
    .word   01300h,00100h
    .word   0d400h,00001h

    .word   00003h,0ffffh     ;Unit_dependent_directory
    .word   04000h,00800h      ;command_regs_base
    .word   08100h,00002h      ;vendor
    .word   08200h,00007h      ;model

    .word   00005h,0ffffh     ;Dir 1, Leaf 1, module_vendor_id text
    .word   00000h,00000h
    .word   00000h,00000h
    .word   04441h,04c53h      ;DALSA INC.
    .word   04120h,0494eh
    .word   0432eh,00000h
```

```
    .word   00005h,0fffbh     ;Dir 1, Leaf 2, model text
    .word   00000h,00000h
    .word   00000h,00000h
    .word   04c49h,04e45h      ;LINE SCAN
    .word   02053h,04341h
    .word   04e00h,00000h

    .word   00000h,00000h

    .newblock

; TSBFIFO: send an isochronous packet out
tsbfifo
    ldp     #grfpage

    mar     *,ar3
    lar     ar3,#08000

$3
    bit     isook,15          ; make sure the iso cycle has started
    bcnd    $5,tc

    banz    $3

    lacl    #023h             ; if not, try to fix it!
    call    debuga

    call    tsbfix

    b       tsbfifo

$5
    lar     ar3,#08000
$2
    setc    intm
    in      tsbl,rinter
    in      tsbh,rinter+1
    clrc    intm

    bit     tsbh,5            ; make sure transmission is available
    bcnd    $6,tc

    banz    $2

    lacl    #024h
    call    debuga

    call    tsbfix            ; if not, try to fix it!

    b       tsbfifo

$6
    setc    intm

    splk    #00fc1h,tsbl      ; set up registers
    splk    #00400h,tsbh

    out     tsbl,rinter
```

```
    out     tsbh,rinter+1

    splk    #isohead,tsbl
    splk    #fpksizeb,tsbh      ; number of bytes (fpksizeb)

$4
    out     tsbl,ritf1
    out     tsbh,ritf1+1

    lar     ar4,#fpksizeq-2         ; block size in quads - 2
    mar     *,ar2

$1
    out     *+,ritf2
    out     *+,ritf2+1,ar4

    banz    $1,ar2

    out     *+,ritf3
    out     *+,ritf3+1

    clrc    intm

    lacl    tempc             ; Print a "." every 2048 packets
    sub     #1
    sacl    tempc
    bcnd    tsbfifoo,neq

    lacc    #2048
    sacl    tempc

    lacl    #02eh
    call    debuga

tsbfifoo
    lacl    #0

tsbfifoe
    ldp     #0
    ret

    .newblock

; TSBFIFOI: initialize ISO tranmission
tsbfifoi
    ldp     #grfpage

    lacc    #2048
    sacl    tempc

    ldp     #0
    ret

    .newblock

; TSBDUMP: dump the contents of the receiver fifo buffer for debugging
tsbdump
    ldp     #grfpage
```

```
tsbdump0
    lacl    grfrp
    and     #000ffh
    or      #grffifo
    sacl    grfrp

    lacl    grfrp
    xor     grfwp
    bcnd    tsbdumpe,eq

    lar     ar2,grfrp
    lacl    grfflag
    call    debugws
    ldp     #grfpage
    mar     *,ar2
    lacl    *+
    call    debugws
    ldp     #grfpage
    mar     *,ar2
    lacl    *+
    call    debugwr

    ldp     #grfpage
    sar     ar2,grfrp

    b       tsbdump0

tsbdumpe
    ldp     #0
    ret

          .newblock

;TSBFIX: try to reset the LLC so that it can continue sending
tsbfix
    setc    intm

    ldp     #grfpage

    splk     #dcontroll,grftsbl          ; Set control registers
    splk     #dcontrolh,grftsbh
    out     grftsbl,rcontrol
    out     grftsbh,rcontrol+1

    lacl    #0
    sacl    isook

    splk    #dintmaskl,grftsbl
    splk    #dintmaskh,grftsbh
    out     grftsbl,rintmask
    out     grftsbh,rintmask+1

    ldp     #0

    clrc    intm

    ret

    .end
```

D.1.4 EEPROM Emulator Utilities

D.1.4.1 loademul.cmd

```
@copy loader.bin/b+prog.bin/b+start.bin/b firmware.bin
@..\bin\urload firmware.cfg
```

D.1.4.2 boot.bin (binary)

Following byte indicates parallel 8 bit bootup mode.

```
$81
```

D.1.4.3 start.bin (binary)

Following bytes are command for firmware loader (3) and location to execute $1000.

```
$03 $10 $00
```

D.1.4.4 firmware.cfg

```
// firmware.cfg
//  script for unirom to load firmware code into emulator

[HOST]
// set interface to parallel port 1
port p378

[cfg]
// set device to 64K FLASH device (with write line)
device  64     0  FLASH
// set reset state
reset   HIGH   TRISTATE
// set arbitration
arbitration c

[cmd]
// reset device
RESET  OFF
// initialize unirom serial communication registers for DSP
fill fff0 fff4 00
// load DSP ROM boot register
load bin boot.bin ffff
// load firmware data
load bin firmware.bin 8000
// set serial communication mode for host
CONSOLE vcom fff0 none 19200
```

D.1.4.5 verify.cmd

```
@copy loader.bin/b+prog.bin/b+start.bin/b firmware.bin
@..\bin\urload verify.cfg
```

D.1.4.6 verify.cfg

```
// verify.cfg
//  script for unirom to verify firmware code in emulator

[HOST]
// set interface to parallel port 1
port p378

[cfg]
// set device to 64K FLASH device (with write line)
device  64     0  FLASH
// set reset state
reset   HIGH   TRISTATE
// set arbitration
arbitration c

[cmd]
// reset device
RESET  OFF
// initialize unirom serial communication registers for DSP
fill fff0 fff4 00
// verify DSP ROM boot register
verify bin boot.bin ffff
// verify firmware data
verify bin final.bin 8000
// set serial communication mode for host
CONSOLE vcom fff0 none 19200
```

D.2 FPGA Hardware Description

D.2.1 Synopsys Scripts

As in the first generation system before Design Compiler can be executed, a setup file (.synopsys_dc.setup) must be located in the current directory which contains the information of the target technology. Once Design Compiler is loaded, a command to load the video processor should be executed: "analyze -f vhdl -lib DALSA <videoprocessor>.vhd". After the code is successfully read in, the automated script can be executed with "include makefpga.scr".

D.2.1.1 .synopsys_dc.setup

```
/* Set information for schematic sheets */
designer = "Roberto Muscedere";
company = "VLSI Research Group";

/* Set synthetic library path, for synopsys libs and alliance libs */
search_path = {. /CMC/tools/synopsys/libraries/syn \
/CMC/tools/xactvm/synopsys/libraries/syn}
/* Set target hardware to 4005-5, include primatives and specific libraries */
link_library = {"*" xprim_4010e-3.db xprim_4000e-3.db xgen_4000e.db xdc_4000e-
3.db xio_4000e-3.db xfpga_4000e-3.db}
target_library = {xprim_4010e-3.db xprim_4000e-3.db xgen_4000e.db xdc_4000e-3.db
xio_4000e-3.db xfpga_4000e-3.db}
/* Add symbol library (for gui interface) */
symbol_library = xc4000e.sdb
/* Add design ware synthetic libraries to improve performance */
define_design_lib xdw_4000e -path \
    /CMC/tools/xactvm/synopsys/libraries/dw/lib/xc4000e
synthetic_library = {xdw_4000e.sldb standard.sldb}

/* Setup outputs for Xilinx XNF file format */
compile_fix_multiple_port_nets = true
xnfout_constraints_per_endpoint = 0
xnfout_library_version = "2.0.0"
bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"
bus_inference_style = "%s<%d>"
edifout_netlist_only = true
edifout_power_and_ground_representation = cell
edifout_write_properties_list = "instance_number port_location part"
xlnx_hier_blknm = 1

/* Define DALSA library location */
define_design_lib DALSA -path "./DALSA"
```

D.2.1.2 makefpga.scr

```
/* be sure to analyze video processor before executing this script */
/* eg:  analyze -f vhdl -lib DALSA processor.vhd */
```

```
/* put everything inside the DALSA library, keeps things neat */
    analyze -f vhdl -lib DALSA commdata.vhd
    analyze -f vhdl -lib DALSA camerasig.vhd
    analyze -f vhdl -lib DALSA reset.vhd
    analyze -f vhdl -lib DALSA fifo.vhd
    analyze -f vhdl -lib DALSA packages.vhd
    analyze -f vhdl -lib DALSA interface.vhd
    elaborate -lib DALSA system_config

/* do not touch special FPGA pins and RAM elements */
    current_design commdata
    set_dont_touch x*

/* do not touch RAM elements */
    current_design system
    set_dont_touch x*

/* set no ports for clock */
    set_pad_type -no_clock "*"
/* except the main clock port and controller clock */
    set_pad_type -clock {CLK}
    set_pad_type -clock {CDCLK}

    set_port_is_pad "*"

/* Set pin locations */
    include fpgapins.scr

/* assign pads to ports */
    insert_pads

/* Set timing constraint to 20 MHz */
    create_clock CLK -period 49 /* 24 */

    compile -map_effort low

/* Not necessary, but nice to see if it is routable */
    report_fpga
/* Not always accurate, but good indication for routability */
    report_timing

    replace_fpga

/* Set FPGA target part number */
    set_attribute system "part" -type string "4010epc84-3"

    set_attribute find(design,"*") "xnfout_use_blknames" -type boolean FALSE
/* Output as sxnf since it contains pin assignment information */
    write -format xnf -hierarchy -output fpga.sxnf

/* Required for new Xilinx Tools and sxnf files */
    write_script > fpga.dc

/* Execute Xilinx Alliance tools to make fpga.stream file */
    sh makefpga
```

D.2.1.3 fpgapins.scr

```
/* Camera lines */

set_attribute {"CLK"} "pad_location" -type string "P13"
set_attribute {"STRB"} "pad_location" -type string "P78"
set_attribute {"LVAL"} "pad_location" -type string "P77"
set_attribute {"USR_EN"} "pad_location" -type string "P5"
set_attribute {"FUTURE"} "pad_location" -type string "P6"
set_attribute {"DEXSYNCIN"} "pad_location" -type string "P23"
set_attribute {"DEXSYNCOUT"} "pad_location" -type string "P24"
set_attribute {"DPRINOUT"} "pad_location" -type string "P70"

set_attribute {"AD<0>"} "pad_location" -type string "P79"
set_attribute {"AD<1>"} "pad_location" -type string "P80"
set_attribute {"AD<2>"} "pad_location" -type string "P81"
set_attribute {"AD<3>"} "pad_location" -type string "P82"
set_attribute {"AD<4>"} "pad_location" -type string "P83"
set_attribute {"AD<5>"} "pad_location" -type string "P84"
set_attribute {"AD<6>"} "pad_location" -type string "P3"
set_attribute {"AD<7>"} "pad_location" -type string "P4"

set_attribute {"BD<0>"} "pad_location" -type string "P7"
set_attribute {"BD<1>"} "pad_location" -type string "P8"
set_attribute {"BD<2>"} "pad_location" -type string "P9"
set_attribute {"BD<3>"} "pad_location" -type string "P10"
set_attribute {"BD<4>"} "pad_location" -type string "P20"
set_attribute {"BD<5>"} "pad_location" -type string "P19"
set_attribute {"BD<6>"} "pad_location" -type string "P18"
set_attribute {"BD<7>"} "pad_location" -type string "P14"

/* FIFO Data lines */

set_attribute {"FOUT<0>"} "pad_location" -type string "P69"
set_attribute {"FOUT<1>"} "pad_location" -type string "P68"
set_attribute {"FOUT<2>"} "pad_location" -type string "P67"
set_attribute {"FOUT<3>"} "pad_location" -type string "P66"
set_attribute {"FOUT<4>"} "pad_location" -type string "P65"
set_attribute {"FOUT<5>"} "pad_location" -type string "P62"
set_attribute {"FOUT<6>"} "pad_location" -type string "P61"
set_attribute {"FOUT<7>"} "pad_location" -type string "P60"
set_attribute {"FOUT<8>"} "pad_location" -type string "P59"

set_attribute {"FOUT<9>"} "pad_location" -type string "P51"
set_attribute {"FOUT<10>"} "pad_location" -type string "P50"
set_attribute {"FOUT<11>"} "pad_location" -type string "P49"
set_attribute {"FOUT<12>"} "pad_location" -type string "P48"
set_attribute {"FOUT<13>"} "pad_location" -type string "P47"
set_attribute {"FOUT<14>"} "pad_location" -type string "P46"
set_attribute {"FOUT<15>"} "pad_location" -type string "P45"
set_attribute {"FOUT<16>"} "pad_location" -type string "P44"
set_attribute {"FOUT<17>"} "pad_location" -type string "P40"

set_attribute {"FOUT<18>"} "pad_location" -type string "P39"
set_attribute {"FOUT<19>"} "pad_location" -type string "P38"
set_attribute {"FOUT<20>"} "pad_location" -type string "P37"
set_attribute {"FOUT<21>"} "pad_location" -type string "P36"
set_attribute {"FOUT<22>"} "pad_location" -type string "P25"
set_attribute {"FOUT<23>"} "pad_location" -type string "P26"
set_attribute {"FOUT<24>"} "pad_location" -type string "P27"
```

```
set_attribute {"FOUT<25>"} "pad_location" -type string "P28"
set_attribute {"FOUT<26>"} "pad_location" -type string "P29"

/* FIFO Control lines */

set_attribute {"WCLK"} "pad_location" -type string "P57"
set_attribute {"WEN1"} "pad_location" -type string "P56"
set_attribute {"WEN2"} "pad_location" -type string "P58"

/* DSP signals */

set_attribute {"CDCLK"} "pad_location" -type string "P35"
set_attribute {"CDIN"} "pad_location" -type string "P71"

set_attribute {"SPARE2"} "pad_location" -type string "P72"
```

D.2.2 Xilinx Scripts

Unlike in the first generation system, Xilinx has obsoleted their XACT software. They have release a new series of tools for synthesizer integration called Alliance. The new Alliance tools operation on all FPGA platforms including the new Spartan and Virtex FPGAs. The command structure is more different than in XACT, but does offer more flexibility. For the second generation system, the Xilinx fields at the beginning of the bitstream must be removed prior to linking into the Windows DLL code. The bitstream contains some information at the beginning of the file describing the target FPGA, date, etc. This data must be removed up to the $FF preamble code; the Unix "tail" is used to do this. The final bitstream is named "fpga.bitstream".

D.2.2.1 makefpga

```
#!/bin/sh

# Change the secondary buffer reference to a primary
ed fpga.sxnf << END
g/BUFG/s/BUFGS/BUFGP/g
w
q
END

# execute all Alliance commands necessary to log file fpga.out
dc2ncf fpga.dc | tee fpga.out
ngdbuild -p xc4010epc84-3 fpga.sxnf | tee -a fpga.out
map fpga | tee -a fpga.out
par fpga -w fpga fpga | tee -a fpga.out
trce fpga -e | tee -a fpga.out
bitgen -w fpga | tee -a fpga.out

# Remove Xilinx text fields from FPGA
```

```
# 66 is for a 4 byte file length, must be adjusted
#  if target is changed from "fpga" to something else
tail +66c fpga.bit >! fpga.stream
```

D.2.3 Main VHDL Code

D.2.3.1 interface.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library GTECH;
use GTECH.GTECH_components.all;

library SYNOPSYS;
use SYNOPSYS.ATTRIBUTES.all;

-- use package descriptions in DALSA package
library DALSA;
use work.DALSA.all;

-- define system ports
entity system is
    port(CLK: in STD_LOGIC;
    FOUT: out STD_LOGIC_VECTOR(26 downto 0);
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    STRB: in STD_LOGIC;
    LVAL: in STD_LOGIC;
    USR_EN: in STD_LOGIC;
    FUTURE: in STD_LOGIC;
    DEXSYNCIN: in STD_LOGIC;
    DEXSYNCOUT: out STD_LOGIC;
    DPRINOUT: out STD_LOGIC;
    CDCLK: in STD_LOGIC;
    CDIN: in STD_LOGIC;
    WEN1: out STD_LOGIC;
    WEN2: out STD_LOGIC;
    WCLK: out STD_LOGIC;
    SPARE2: out STD_LOGIC );
end system;

architecture behaviour of system is

constant CYCLE1: STD_LOGIC := '0';
constant CYCLE2: STD_LOGIC := '1';

-- define gluing signals
signal tomdlpin: STD_LOGIC;
signal tomdlbuf: STD_LOGIC;
signal totdopin: STD_LOGIC;
signal totdobuf: STD_LOGIC;
signal totmspin: STD_LOGIC;
signal totmsbuf: STD_LOGIC;
signal fromtckpin: STD_LOGIC;
```

```
signal fromtckbuf: STD_LOGIC;
signal fromtdipin: STD_LOGIC;
signal fromtdibuf: STD_LOGIC;

signal CDOUT: STD_LOGIC;
signal FRS: STD_LOGIC;
signal PAF: STD_LOGIC;
signal PAE: STD_LOGIC;

signal CD_VAR0: UNSIGNED(7 downto 0);
signal CD_VAR1: UNSIGNED(7 downto 0);
signal CD_VAR2: UNSIGNED(7 downto 0);
signal CD_VAR3: UNSIGNED(7 downto 0);
signal CD_VAR4: UNSIGNED(7 downto 0);
signal CD_VAR5: UNSIGNED(7 downto 0);
signal CD_VAR6: UNSIGNED(7 downto 0);
signal CD_VAR7: UNSIGNED(7 downto 0);
signal CD_REGO: UNSIGNED(7 downto 0);
signal CD_REGI: UNSIGNED(7 downto 0);
signal CD_BITRD: STD_LOGIC_VECTOR(15 downto 0);
signal CD_BITRA: STD_LOGIC_VECTOR(3 downto 0);

signal R_RESET: STD_LOGIC;

signal PR_PUSH: STD_LOGIC;
signal PR_OUTDATA: STD_LOGIC_VECTOR(26 downto 0);

signal LOGIC0: STD_LOGIC;
signal LOGIC1: STD_LOGIC;

signal SPARE: STD_LOGIC;

begin

-- Instantiate special FPGA components
    xMD1BUF: OBUF port map ( I => tomd1buf , O => tomd1pin );
    xMD1PIN: MD1 port map ( O => tomd1pin );
    tomd1buf <= CDOUT;

    xTDOBUF: OBUF port map ( I => totdobuf , O => totdopin );
    xTDOPIN: TDO port map ( O => totdopin );
    totdobuf <= FRS;

    xTCKPIN: TCK port map ( I => fromtckpin );
    xTCKBUF: IBUF port map ( I => fromtckpin , O => fromtckbuf );
    PAF <= fromtckbuf;

    xTDIPIN: TDI port map ( I => fromtdipin );
    xTDIBUF: IBUF port map ( I => fromtdipin , O => fromtdibuf );
    PAE <= fromtdibuf;

    LOGIC1 <= '1';
    LOGIC0 <= '0';

    SPARE2 <= '0';

-- Instantiate sub-sys components with connectivity
    s_reset: reset port map ( CLK => CLK, RESET => R_RESET );
```

```
    s_fifo: fifo port map ( CLK => CLK,  -- READ => CD_REGO(1),
        RESETFIFO => CD_REGO(0), WRITE => PR_PUSH, DATAIN => PR_OUTDATA,
        DATAOUT => FOUT, WEN1 => WEN1, WEN2 => WEN2, WCLK => WCLK,
        RS => FRS, RESET => R_RESET );

    s_processor: processor port map (CLK => CLK, RESET => R_RESET, AD => AD,
        BD => BD, LVAL => LVAL, PVAL => STRB, OUTDATA => PR_OUTDATA,
        PUSH => PR_PUSH, CAPTUREON => CD_REGO(2), OKTOPUSH => PAF,
        BITRD => CD_BITRD, BITRA => CD_BITRA,
        VAR0 => CD_VAR0, VAR1 => CD_VAR1, VAR2 => CD_VAR2, VAR3 => CD_VAR3,
        VAR4 => CD_VAR4, VAR5 => CD_VAR5, VAR6 => CD_VAR6, VAR7 => CD_VAR7 );

    s_camerasig: camerasig port map (CLK => CLK, DEXSYNCIN => DEXSYNCIN,
        DEXSYNCOUT => DEXSYNCOUT, DPRINOUT => DPRINOUT);

    s_commdata: commdata port map (CDCLK => CDCLK, CDIN => CDIN, CDOUT => CDOUT,
        RESET => R_RESET, BITRD => CD_BITRD, BITRA => CD_BITRA,
        VAR0 => CD_VAR0, VAR1 => CD_VAR1, VAR2 => CD_VAR2, VAR3 => CD_VAR3,
        VAR4 => CD_VAR4, VAR5 => CD_VAR5, VAR6 => CD_VAR6, VAR7 => CD_VAR7,
        REGO => CD_REGO, REGI => CD_REGI );
-- use this for register loop back tests
--      REGO => CD_REGO, REGI => CD_REGO );

end behaviour;

-- use this for debugging with vhdldbx
configuration system_config of system is
    for behaviour
        for s_reset: reset use entity DALSA.reset(behaviour);
        end for;
        for s_fifo: fifo use entity DALSA.fifo(behaviour);
        end for;
        for s_processor: processor use entity DALSA.processor(behaviour);
        end for;
        for s_camerasig: camerasig use entity DALSA.camerasig(behaviour);
        end for;
        for s_commdata: commdata use entity DALSA.commdata(behaviour);
        end for;
    end for;
end system_config;
```

D.2.3.2 packages.vhd

```
-- Inlcude necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library GTECH;
use GTECH.GTECH_components.all;

library SYNOPSYS;
use SYNOPSYS.ATTRIBUTES.all;

-- define all components into a single package
package DALSA is

-- special XC4000 ports
```

```
component MD0
    port(I: out STD_LOGIC );
end component;

component MD1
    port(O: in STD_LOGIC );
end component;

component MD2
    port(I: out STD_LOGIC );
end component;

component TCK
    port(I: out STD_LOGIC );
end component;

component TDI
    port(I: out STD_LOGIC );
end component;

component TMS
    port(I: out STD_LOGIC );
end component;

component TDO
    port(O: in STD_LOGIC );
end component;

component IBUF
    port(I: in STD_LOGIC;
    O: out STD_LOGIC );
end component;

component OBUF
    port(I: in STD_LOGIC;
    O: out STD_LOGIC );
end component;

-- Sub-system definitions
component processor
port(CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;
    BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
```

```
    OKTOPUSH: in STD_LOGIC );
end component;

component camerasig
    port(CLK: in STD_LOGIC;
    DEXSYNCIN: in STD_LOGIC;
    DEXSYNCOUT: out STD_LOGIC;
    DPRINOUT: out STD_LOGIC );
end component;

component commdata
port(CDCLK: in STD_LOGIC;
    CDIN: in STD_LOGIC;
    CDOUT: out STD_LOGIC;
    RESET: in STD_LOGIC;
    BITRD: out STD_LOGIC_VECTOR(15 downto 0);
    BITRA: in STD_LOGIC_VECTOR(3 downto 0);
    VAR0: out UNSIGNED(7 downto 0);
    VAR1: out UNSIGNED(7 downto 0);
    VAR2: out UNSIGNED(7 downto 0);
    VAR3: out UNSIGNED(7 downto 0);
    VAR4: out UNSIGNED(7 downto 0);
    VAR5: out UNSIGNED(7 downto 0);
    VAR6: out UNSIGNED(7 downto 0);
    VAR7: out UNSIGNED(7 downto 0);
    REGO: out UNSIGNED(7 downto 0);
    REGI: in UNSIGNED(7 downto 0) );
end component;

component fifo
    port(CLK: in STD_LOGIC;
    RESETFIFO: in STD_LOGIC;
    WRITE: in STD_LOGIC;
    DATAIN: in STD_LOGIC_VECTOR(26 downto 0);
    DATAOUT: out STD_LOGIC_VECTOR(26 downto 0);
    WEN1: out STD_LOGIC;
    WEN2: out STD_LOGIC;
    WCLK: out STD_LOGIC;
    RS: out STD_LOGIC;
    RESET: in STD_LOGIC );
end component;

component reset
    port(CLK: in STD_LOGIC;
    RESET: out STD_LOGIC );
end component;

end DALSA;
```

D.2.3.3 reset.vhd

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define sub-system ports
entity reset is
```

```
    port(CLK: in STD_LOGIC;
    RESET: out STD_LOGIC );
end reset;

architecture behaviour of reset is
signal COUNT: STD_LOGIC_VECTOR(1 downto 0);
begin

    main: process(CLK)
    begin

-- use a counter of 3
    if (CLK'event and CLK='0') then

-- when counter reaches 3, set all asynchronous resets
        if (COUNT="11") then
            RESET <= '1';
        else
-- else increment 2 bit counter
            COUNT <= COUNT + 1;
            RESET <= '0';
        end if;

    end if;

    end process;

end behaviour;
```

D.2.3.4 camerasig.vhd

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define sub-system ports
entity camerasig is
    port(CLK: in STD_LOGIC;
    DEXSYNCIN: in STD_LOGIC;
    DEXSYNCOUT: out STD_LOGIC;
    DPRINOUT: out STD_LOGIC );
end camerasig;

architecture behaviour of camerasig is
begin

    main: process(DEXSYNCIN)
    begin

-- nothing to do with the signals rightnow, just pass them through
    DEXSYNCOUT <= DEXSYNCIN;
    DPRINOUT <= '1';

    end process;

end behaviour;
```

D.2.3.5 commdata.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library GTECH;
use GTECH.GTECH_components.all;

-- define sub-system ports
entity commdata is
    port(CDCLK: in STD_LOGIC;
    CDIN: in STD_LOGIC;
    CDOUT: out STD_LOGIC;
    RESET: in STD_LOGIC;
    BITRD: out STD_LOGIC_VECTOR(15 downto 0);
    BITRA: in STD_LOGIC_VECTOR(3 downto 0);
    VAR0: out UNSIGNED(7 downto 0);
    VAR1: out UNSIGNED(7 downto 0);
    VAR2: out UNSIGNED(7 downto 0);
    VAR3: out UNSIGNED(7 downto 0);
    VAR4: out UNSIGNED(7 downto 0);
    VAR5: out UNSIGNED(7 downto 0);
    VAR6: out UNSIGNED(7 downto 0);
    VAR7: out UNSIGNED(7 downto 0);
    REGO: out UNSIGNED(7 downto 0);
    REGI: in UNSIGNED(7 downto 0) );
end commdata;

architecture behaviour of commdata is

-- define special Xilinx XC4000E Dual Port 16x1 component
component RAMD
    port(A0: in STD_LOGIC;
    A1: in STD_LOGIC;
    A2: in STD_LOGIC;
    A3: in STD_LOGIC;
    DPRA0: in STD_LOGIC;
    DPRA1: in STD_LOGIC;
    DPRA2: in STD_LOGIC;
    DPRA3: in STD_LOGIC;
    WE: in STD_LOGIC;
    WCLK: in STD_LOGIC;
    D: in STD_LOGIC;
    DPO: out STD_LOGIC );
end component;

-- create an array for the programmable parameters
type rammemory is array(7 downto 0) of UNSIGNED(7 downto 0);

signal RXSTATE: STD_LOGIC_VECTOR(3 downto 0);
signal RXBUF: UNSIGNED(12 downto 0);
signal WXSTATE: STD_LOGIC_VECTOR(2 downto 0);
signal WEXECUTE: STD_LOGIC;
signal REXECUTE: STD_LOGIC;
signal RAM: rammemory;

signal BITWADDR: STD_LOGIC_VECTOR(3 downto 0);
```

```
signal BITWDATA: STD_LOGIC_VECTOR(15 downto 0);
signal BITWE: STD_LOGIC;

begin

-- set ports to the paramater array
    setting: process(RAM)
    begin

    VAR0 <= RAM(0);
    VAR1 <= RAM(1);
    VAR2 <= RAM(2);
    VAR3 <= RAM(3);
    VAR4 <= RAM(4);
    VAR5 <= RAM(5);
    VAR6 <= RAM(6);
    VAR7 <= RAM(7);

    end process;

-- process an input serial stream from the controller
    readin: process(RESET,CDCLK)
    begin

-- handle reset
    if (RESET='0') then

        RXSTATE <= "0000";
        WEXECUTE <= '0';

    elsif (CDCLK'event and CDCLK='1') then

        case RXSTATE is

            when "0000" => -- start bit "must be 0"

                if (CDIN = '1') then
                    RXSTATE <= "0000";
                else
                    RXSTATE <= "0001";
                end if;
                WEXECUTE <= '0';

            when "0001" => -- bit 1

                RXBUF(12 downto 1) <= RXBUF(11 downto 0);
                RXBUF(0) <= CDIN;
                RXSTATE <= "0010";
                WEXECUTE <= '0';

            when "0010" => -- bit 2

                RXBUF(12 downto 1) <= RXBUF(11 downto 0);
                RXBUF(0) <= CDIN;
                RXSTATE <= "0011";
                WEXECUTE <= '0';

            when "0011" => -- bit 3
```

```
        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "0100";
        WEXECUTE <= '0';

    when "0100" => -- bit 4

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "0101";
        WEXECUTE <= '0';

    when "0101" => -- bit 5

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "0110";
        WEXECUTE <= '0';

    when "0110" => -- bit 6

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "0111";
        WEXECUTE <= '0';

    when "0111" => -- bit 7

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "1000";
        WEXECUTE <= '0';

    when "1000" => -- bit 8

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "1001";
        WEXECUTE <= '0';

    when "1001" => -- bit 9

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "1010";
        WEXECUTE <= '0';

    when "1010" => -- bit 10

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "1011";
        WEXECUTE <= '0';

    when "1011" => -- bit 11

        RXBUF(12 downto 1) <= RXBUF(11 downto 0);
        RXBUF(0) <= CDIN;
        RXSTATE <= "1100";
```

```
            WEXECUTE <= '0';

         when "1100" => -- bit 12

            RXBUF(12 downto 1) <= RXBUF(11 downto 0);
            RXBUF(0) <= CDIN;
            RXSTATE <= "1101";
            WEXECUTE <= '0';

         when "1101" => -- bit 13

            RXBUF(12 downto 1) <= RXBUF(11 downto 0);
            RXBUF(0) <= CDIN;
            RXSTATE <= "1111";
            WEXECUTE <= '0';

         when "1111" => -- stop bit must be "1"

            if (CDIN='1') then
               WEXECUTE <= '1';
            else
               WEXECUTE <= '0';
            end if;
            RXSTATE <= "0000";

         when others =>

            NULL;

      end case;
   end if;

   end process;

-- process register and ram operations
   executing: process(RESET,CDCLK)
   begin

-- handle reset
   if (RESET='0') then

      REXECUTE <= '0';

   elsif (CDCLK'event and CDCLK='1') then

-- write event
      if (WEXECUTE = '1') then
         if (RXBUF(12) = '1') then -- write operation
            if (RXBUF(11) = '1') then -- to ram space
               if (RXBUF(10) = '0') then
                  case RXBUF(9 downto 8) is
                     when "11" => -- set address and write to RAM
                        BITWADDR <= CONV_STD_LOGIC_VECTOR(RXBUF(3 downto
0),4);
                        BITWE <= '1';
                     when others => -- do nothing
                        REGO <= RXBUF(7 downto 0);
                        BITWE <= '0';
                  end case;
```

```
                else
                    BITWE <= '0'; -- do not write to registers
                    case RXBUF(9 downto 8) is
                        when "00" | "10" => -- set low ram data
                            BITWDATA(7 downto 0) <= CONV_STD_LOGIC_VECTOR(RXBUF(7
downto 0),8);
                        when "01" | "11" => -- set high ram data
                            BITWDATA(15 downto 8) <=
CONV_STD_LOGIC_VECTOR(RXBUF(7 downto 0),8);
                        when others =>
                            NULL;
                    end case;
                end if;
            else -- to register space
                BITWE <= '0'; -- do not write to RAM
                RAM(CONV_INTEGER(RXBUF(10 downto 8))) <= RXBUF(7 downto 0); --
write to address with data
            end if;
            REXECUTE <= '0';
        else -- read operation
-- exclude to optimize out RAM if it is not needed
            BITWE <= '0';
--          if (RXBUF(11) = '1') then -- from algorithm maybe if it has
anything to say
--              WXBUF <= REGI;
--          else -- from register space
--              WXBUF <= RAM(CONV_INTEGER(RXBUF(10 downto 8)));
--          end if;
            REXECUTE <= '1';
        end if;
    else
-- no command, do nothing
        BITWE <= '0';
        REXECUTE <= '0';
    end if;

end if;

end process;

-- send a serial data stream to the FPGA for read
    writeout: process(RESET,CDCLK)
    begin

-- handle reset
    if (RESET='0') then
        WXSTATE <= "000";
        CDOUT <= '1';

    elsif (CDCLK'event and CDCLK='1') then

-- send the bits
        case WXSTATE is

            when "000" => -- bit 1

                if (REXECUTE = '0') then
                    WXSTATE <= "000";
                else
```

```
                WXSTATE <= "001";
            end if;
            CDOUT <= REGI(7);

        when "001" => -- bit 2

            CDOUT <= REGI(6);
            WXSTATE <= "010";

        when "010" => -- bit 3

            CDOUT <= REGI(5);
            WXSTATE <= "011";

        when "011" => -- bit 4

            CDOUT <= REGI(4);
            WXSTATE <= "100";

        when "100" => -- bit 5

            CDOUT <= REGI(3);
            WXSTATE <= "101";

        when "101" => -- bit 6

            CDOUT <= REGI(2);
            WXSTATE <= "110";

        when "110" => -- bit 7

            CDOUT <= REGI(1);
            WXSTATE <= "111";

        when "111" => -- bit 8

            CDOUT <= REGI(0);
            WXSTATE <= "000";

        when others =>

            NULL;

    end case;

end if;

end process;

-- Instantiate RAM components

    xdram00: RAMD port map ( D => BITWDATA(0), DPO => BITRD(0),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

    xdram01: RAMD port map ( D => BITWDATA(1), DPO => BITRD(1),
```

```
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram02: RAMD port map ( D => BITWDATA(2), DPO => BITRD(2),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram03: RAMD port map ( D => BITWDATA(3), DPO => BITRD(3),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram04: RAMD port map ( D => BITWDATA(4), DPO => BITRD(4),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram05: RAMD port map ( D => BITWDATA(5), DPO => BITRD(5),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram06: RAMD port map ( D => BITWDATA(6), DPO => BITRD(6),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram07: RAMD port map ( D => BITWDATA(7), DPO => BITRD(7),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram08: RAMD port map ( D => BITWDATA(8), DPO => BITRD(8),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
      WCLK => CDCLK, WE => BITWE );

   xdram09: RAMD port map ( D => BITWDATA(9), DPO => BITRD(9),
      A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
      DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
```

```
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

   xdram10: RAMD port map ( D => BITWDATA(10), DPO => BITRD(10),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

   xdram11: RAMD port map ( D => BITWDATA(11), DPO => BITRD(11),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

   xdram12: RAMD port map ( D => BITWDATA(12), DPO => BITRD(12),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

   xdram13: RAMD port map ( D => BITWDATA(13), DPO => BITRD(13),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

   xdram14: RAMD port map ( D => BITWDATA(14), DPO => BITRD(14),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

   xdram15: RAMD port map ( D => BITWDATA(15), DPO => BITRD(15),
       A0 => BITWADDR(0), A1 => BITWADDR(1), A2 => BITWADDR(2), A3 =>
BITWADDR(3),
       DPRA0 => BITRA(0), DPRA1 => BITRA(1), DPRA2 => BITRA(2), DPRA3 =>
BITRA(3),
       WCLK => CDCLK, WE => BITWE );

end behaviour;
```

D.2.3.6 fifo.vhd

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define sub-system ports
entity fifo is
   port(CLK: in STD_LOGIC;
   RESETFIFO: in STD_LOGIC;
```

```
    WRITE: in STD_LOGIC;
    DATAIN: in STD_LOGIC_VECTOR(26 downto 0);
    DATAOUT: out STD_LOGIC_VECTOR(26 downto 0);
    WEN1: out STD_LOGIC;
    WEN2: out STD_LOGIC;
    WCLK: out STD_LOGIC;
    RS: out STD_LOGIC;
    RESET: in STD_LOGIC );
end fifo;

architecture behaviour of fifo is

-- define an array for almost flag values
subtype RS_WORD is STD_LOGIC_VECTOR(7 downto 0);
subtype RS_RANGE is INTEGER range 0 to 3;
type RS_TEMPLATE is array (0 to RS_RANGE'high) of RS_WORD;

constant RS_TABLE: RS_TEMPLATE := (
    "00000000","10000000","00001000","00100000"
);

signal DATA1: STD_LOGIC_VECTOR(26 downto 0);
signal DATA2: STD_LOGIC_VECTOR(26 downto 0);
signal HOLD: STD_LOGIC_VECTOR(26 downto 0);
signal STATE: STD_LOGIC_VECTOR(2 downto 0);
signal INITS: STD_LOGIC_VECTOR(7 downto 0);
signal RSLINES: STD_LOGIC;
signal LRESET: STD_LOGIC;
begin

    clocking: process(CLK)
    begin

-- write clock set from master
        WCLK <= CLK;

    end process;

    main: process(RESET,CLK)
    begin

-- on system reset do nothing to FIFO
        if (RESET = '0') then

            RS <= '1';
            RSLINES <= '0';
            STATE <= (others => '0');
            WEN1 <= '1';
            WEN2 <= '1';

        elsif (CLK'event and CLK='0') then

            LRESET <= RESETFIFO;

            case STATE is

                when "000" =>

                    INITS <= RS_TABLE(0);
```

```
                if (RESETFIFO='1' and LRESET='0') then
-- reset FIFO, assert reset line (stage 1)
                    RS <= '0';
                    RSLINES <= '1';
                    STATE <= "001";
                    WEN1 <= '1';
                    WEN2 <= '0';
                else
-- do nothing to FIFO
                    RS <= '1';
                    RSLINES <= '0';
                    STATE <= "000";
                    WEN1 <= not WRITE;
                    WEN2 <= '1';
                    HOLD <= DATAIN;
                end if;

            when "001" =>

-- reset FIFO, assert reset line (stage 2)
                INITS <= RS_TABLE(0);
                RS <= '0';
                RSLINES <= '1';
                STATE <= "010";
                WEN1 <= '1';
                WEN2 <= '0';

            when "010" =>

-- reset FIFO, wait (stage 3)
                INITS <= RS_TABLE(0);
                RS <= '1';
                RSLINES <= '1';
                STATE <= "011";
                WEN1 <= '1';
                WEN2 <= '0';

            when "011" =>

-- reset FIFO, wait (stage 4)
                INITS <= RS_TABLE(0);
                RS <= '1';
                RSLINES <= '0';
                STATE <= "100";
                WEN1 <= '1';
                WEN2 <= '1';

            when "100" =>

-- reset FIFO, program almost empty MSB (stage 5)
                INITS <= RS_TABLE(0);
                RS <= '1';
                RSLINES <= '1';
                STATE <= "101";
                WEN1 <= '0';
                WEN2 <= '0';

            when "101" =>
```

```
-- reset FIFO, program almost empty LSB  (stage 6)
                INITS <= RS_TABLE(1);
                RS <= '1';
                RSLINES <= '1';
                STATE <= "110";
                WEN1 <= '0';
                WEN2 <= '0';

            when "110" =>

-- reset FIFO, program almost full MSB (stage 7)
                INITS <= RS_TABLE(2);
                RS <= '1';
                RSLINES <= '1';
                STATE <= "111";
                WEN1 <= '0';
                WEN2 <= '0';

            when "111" =>

-- reset FIFO,  program almost full MSB (stage 8)
                INITS <= RS_TABLE(3);
                RS <= '1';
                RSLINES <= '1';
                STATE <= "000";
                WEN1 <= '0';
                WEN2 <= '0';

            when others =>

                NULL;

        end case;

    end if;

end process;

-- when in FIFO reset mode, use tristates to multiplex reset data
    linecontrol: process(RSLINES,STATE,HOLD,INITS)
    begin

        if (RSLINES = '1') then
            DATA1 <= INITS & '0' & INITS & '0' & INITS & '0';
            DATA2 <= (others => 'Z');
        else
            DATA1 <= (others => 'Z');
            DATA2 <= HOLD;
        end if;

    end process;

-- the way we do a wire-and
    lineset1: process(DATA1)
    begin

        DATAOUT <= DATA1;
```

```
    end process;

    lineset2: process(DATA2)
    begin

        DATAOUT <= DATA2;

    end process;

end behaviour;
```

D.2.4 Video Processor VHDL Code

D.2.4.1 focus.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    FVAL: in STD_LOGIC;
    BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal FIRSTPIXEL: STD_LOGIC;
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal FVALSIG: STD_LOGIC;
signal TOBESYNCED: STD_LOGIC;
signal CURRENT: UNSIGNED(7 downto 0);
signal COUNTER: UNSIGNED(10 downto 0);
signal TEMP0: UNSIGNED(10 downto 0);
signal TEMP1: UNSIGNED(10 downto 0);
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);
begin
```

```
-- convert programmable parameters into something we can use
    TEMP0 <= VAR2(2 downto 0) & VAR0;
    TEMP1 <= VAR2(5 downto 3) & VAR1;

-- stabalize camera inputs
    camera_proc1: process begin
    wait until CLK'event and CLK='0';

        CURRENT <= AD;
        LVALSIG <= LVAL;
        FVALSIG <= FVAL;

    end process camera_proc1;

    camera_proc3: process begin
        wait until CLK'event and CLK='0';

-- when pixel is valid
        if (FVALSIG='0' and CAPTUREON='1') then

            OUTDATA(25 downto 24) <= (others => '0');
-- set sync line
            OUTDATA(26) <= TOBESYNCED;

-- when line is valid
            if (LVALSIG='1') then

-- initialize variable on first pixel
                if ((FIRSTPIXEL='1' and OKTOPUSH='0') or PROCLINE='0') then

                    PROCLINE <= '0';
                    INDEX <= "00";
                    PUSH <= '0';
                    TOBESYNCED <= '0';
                    FIRSTPIXEL <= '0';

                else

-- increment pixel counter
                    COUNTER <= COUNTER + 1;

-- check for range for focus
                    if ( COUNTER >= TEMP0 and COUNTER <= TEMP1 ) then

                        FIRSTPIXEL <= '0';
                        PROCLINE <= '1';

-- figure out which of 3 8-bit sections to write to
                        case INDEX is
                            when "00" =>
                                OUTDATA(23 downto 16) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                                INDEX <= "01";
                                PUSH <= '0';
                            when "01" =>
                                OUTDATA(15 downto 8) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                                INDEX <= "10";
```

```
                        PUSH <= '0';
                    when "10" =>
                        OUTDATA(7 downto 0) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                        INDEX <= "00";
-- on the last one, write the 27-bit data to the FIFO
                        PUSH <= '1';
                        TOBESYNCED <= '0';
                    when others =>
                        NULL;
                end case;

            else

                FIRSTPIXEL <= '0';
                PROCLINE <= '1';
                PUSH <= '0';

            end if;

        end if;

    else

-- If we didn't write all the data, do it now
        if ( (not (INDEX = "00")) and TOBESYNCED='0' ) then
            PUSH <= '1';
        else
            PUSH <= '0';
        end if;
-- get ready for next line
        INDEX <= "00";
        TOBESYNCED <= '1';
        FIRSTPIXEL <= '1';
        PROCLINE <= '1';
        COUNTER <= (others => '0');

    end if;
  else
-- nothing to be done, don't write
    PUSH <= '0';
  end if;

end process camera_proc3;

end behaviour;
```

D.2.4.2 lineup.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(CLK: in STD_LOGIC;
   RESET: in STD_LOGIC;
```

```
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;
    BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal FIRSTPIXEL: STD_LOGIC;
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal TOBESYNCED: STD_LOGIC;
signal CURRENT: UNSIGNED(7 downto 0);
signal COUNTER: UNSIGNED(10 downto 0);
signal TEMP0: UNSIGNED(10 downto 0);
signal TEMP1: UNSIGNED(10 downto 0);
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);
begin

-- convert programmable parameters into something we can use
    TEMP0 <= VAR2(2 downto 0) & VAR0;
    TEMP1 <= VAR2(5 downto 3) & VAR1;

-- stabalize camera inputs
    camera_proc1: process begin
        wait until CLK'event and CLK='0';

        CURRENT <= AD;
        LVALSIG <= LVAL;
        PVALSIG <= PVAL;

    end process camera_proc1;

    camera_proc3: process begin
        wait until CLK'event and CLK='0';

-- when pixel is valid
        if (PVALSIG='0' and CAPTUREON='1') then

            OUTDATA(25 downto 24) <= (others => '0');
-- set sync line
            OUTDATA(26) <= TOBESYNCED;

-- when line is valid
```

```
          if (LVALSIG='1') then

-- initialize variable on first pixel
              if ((FIRSTPIXEL='1' and OKTOPUSH='0') or PROCLINE='0') then

                  PROCLINE <= '0';
                  INDEX <= "00";
                  PUSH <= '0';
                  TOBESYNCED <= '0';
                  FIRSTPIXEL <= '0';

              else

-- increment pixel counter
                  COUNTER <= COUNTER + 1;

-- check for range for lineup
                  if ( COUNTER < TEMP0 or COUNTER > TEMP1 ) then

                      FIRSTPIXEL <= '0';
                      PROCLINE <= '1';

-- figure out which of 3 8-bit sections to write to
                      case INDEX is
                          when "00" =>
                              OUTDATA(23 downto 16) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                              INDEX <= "01";
                              PUSH <= '0';
                          when "01" =>
                              OUTDATA(15 downto 8) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                              INDEX <= "10";
                              PUSH <= '0';
                          when "10" =>
                              OUTDATA(7 downto 0) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                              INDEX <= "00";
-- on the last one, write the 27-bit data to the FIFO
                              PUSH <= '1';
                              TOBESYNCED <= '0';
                          when others =>
                              NULL;
                      end case;

                  else

                  FIRSTPIXEL <= '0';
                  PROCLINE <= '1';
                  PUSH <= '0';

                  end if;

              end if;

          else

-- If we didn't write all the data, do it now
              if ( (not (INDEX = "00")) and TOBESYNCED='0' ) then
```

```
                PUSH <= '1';
            else
                PUSH <= '0';
            end if;
-- get ready for next line
            INDEX <= "00";
            TOBESYNCED <= '1';
            FIRSTPIXEL <= '1';
            PROCLINE <= '1';
            COUNTER <= (others => '0');

        end if;
    else
-- nothing to be done, don't write
        PUSH <= '0';
    end if;

  end process camera_proc3;

end behaviour;
```

D.2.4.3 minmax.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(CLK: in STD_LOGIC;
  RESET: in STD_LOGIC;
  AD: in UNSIGNED(7 downto 0);
  BD: in UNSIGNED(7 downto 0);
  LVAL: in STD_LOGIC;
  PVAL: in STD_LOGIC;
  BITRD: in STD_LOGIC_VECTOR(15 downto 0);
  BITRA: out STD_LOGIC_VECTOR(3 downto 0);
  VAR0: in UNSIGNED(7 downto 0);
  VAR1: in UNSIGNED(7 downto 0);
  VAR2: in UNSIGNED(7 downto 0);
  VAR3: in UNSIGNED(7 downto 0);
  VAR4: in UNSIGNED(7 downto 0);
  VAR5: in UNSIGNED(7 downto 0);
  VAR6: in UNSIGNED(7 downto 0);
  VAR7: in UNSIGNED(7 downto 0);
  OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
  PUSH: out STD_LOGIC;
  CAPTUREON: in STD_LOGIC;
  OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal FIRSTPIXEL: STD_LOGIC;
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
```

```
signal TOBESYNCED: STD_LOGIC;
signal CURRENT: UNSIGNED(7 downto 0);
signal DATA: STD_LOGIC_VECTOR(7 downto 0);
signal PUSHED: STD_LOGIC_VECTOR(17 downto 0);
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);
signal COUNT: STD_LOGIC_VECTOR(7 downto 0);
signal CPUSH: STD_LOGIC;
signal CCOMP: STD_LOGIC;
signal CEND: STD_LOGIC;
signal CLCOMP: STD_LOGIC;
signal CPROC: STD_LOGIC;
signal CCONT: STD_LOGIC;
signal WIDE: STD_LOGIC;
signal COUNTER: UNSIGNED(10 downto 0);
signal TEMP0: UNSIGNED(10 downto 0);
signal TEMP1: UNSIGNED(10 downto 0);
begin

-- convert programmable parameters into something we can use
   TEMP0 <= VAR2(2 downto 0) & VAR0;
   TEMP1 <= VAR2(5 downto 3) & VAR1;

-- stabalize camera inputs
   camera_proc1: process begin
   wait until CLK'event and CLK='0';

       CURRENT <= AD;
       LVALSIG <= LVAL;
       PVALSIG <= PVAL;

   end process camera_proc1;

   camera_proc2: process begin
       wait until CLK'event and CLK='0';

-- when pixel is valid
       if (PVALSIG='0' and CAPTUREON='1') then

-- when line is valid
          if (LVALSIG='1') then

              TOBESYNCED <= '1';

-- initialize variable on first pixel
              if ((FIRSTPIXEL='1' and OKTOPUSH='0') or PROCLINE='0') then

                  FIRSTPIXEL <= '0';
                  PROCLINE <= '0';
                  CPROC <= '0';
                  CCOMP <= '1';
                  CEND <= '0';

              else

                  FIRSTPIXEL <= '0';
                  PROCLINE <= '1';
                  CEND <= '0';

-- increment pixel counter
```

```
                COUNTER <= COUNTER + 1;

-- check for range
                if ( COUNTER >= TEMP0 and COUNTER <= TEMP1 ) then

                    CPROC <= '1';

                    if ( (CURRENT <= VAR3) or (CURRENT >= VAR4) ) then

-- defect pixel
                        DATA <= CONV_STD_LOGIC_VECTOR(CURRENT,8);
                        CCOMP <= '0';

                    else

-- non-defect pixel
                        CCOMP <= '1';

                    end if;

                else

-- nothing, not in range
                    CPROC <= '0';

                end if;

            end if;

        else

            CCOMP <= '0';
            CEND <= '1';

-- process end of line if necessary
            if ( TOBESYNCED='1' and PROCLINE='1' ) then
                CPROC <= '1';
            else
                CPROC <= '0';
            end if;

-- get ready for next line
            TOBESYNCED <= '0';
            FIRSTPIXEL <= '1';
            PROCLINE <= '1';
            COUNTER <= ( others => '0' );

        end if;
    else
        CPROC <= '0';
        CCOMP <= '1';
        CEND <= '0';
    end if;

end process camera_proc2;

-- Perform simple RLE compression
camera_proc3: process
variable CASEVAR: STD_LOGIC_VECTOR(2 downto 0);
```

```
   begin

      wait until CLK'event and CLK='0';

      CASEVAR := CCOMP & CLCOMP & CEND;

-- do it on data to be processed
      if (CPROC='1') then

         case CASEVAR is

            when "011" | "111" => -- end of line, send sync and last repeat

               PUSHED <= '1' & "00000000" & '1' & COUNT;
               WIDE <= '1';
               CPUSH <= '1';
               CLCOMP <= '0';

            when "001" | "101" => -- end of line, send sync

               PUSHED(8 downto 0) <= '1' & "00000000";
               WIDE <= '0';
               CPUSH <= '1';
               CLCOMP <= '0';

            when "110" => -- repeat data, increment counter, dump if overflow

               if (COUNT="11111111") then
                  PUSHED(8 downto 0) <= '1' & COUNT;
                  WIDE <= '0';
                  CPUSH <= '1';
                  CLCOMP <= '1';
                  COUNT <= "00000001";
               else
                  COUNT <= COUNT + 1;
                  WIDE <= '0';
                  CPUSH <= '0';
                  CLCOMP <= '1';
               end if;

            when "100" => -- start of RLE encoding

               COUNT <= "00000001";
               WIDE <= '0';
               CPUSH <= '0';
               CLCOMP <= '1';

            when "010" => -- send luminance and repeat of last value

               PUSHED <= '0' & DATA & '1' & COUNT;
               WIDE <= '1';
               CPUSH <= '1';
               CLCOMP <= '0';

            when "000" => -- send luminance only

               PUSHED(8 downto 0) <= '0' & DATA;
               WIDE <= '0';
               CPUSH <= '1';
```

```
                CLCOMP <= '0';

            when others => -- nothing

                NULL;

        end case;

    else

        CPUSH <= '0';

    end if;

end process camera_proc3;

-- Dump RLE block to FIFO
    camera_proc4: process begin

        wait until CLK'event and CLK='0';

-- Push 16 bit values in 2 cycles
        if (CPUSH='1' or CCONT='1') then

            if (CPUSH='1' and WIDE='1') then
                CCONT <= '1';
            else
                CCONT <= '0';
            end if;

            if (CCONT = '1') then
-- second cycle (luminance or sync)
                case INDEX is
                    when "00" =>
                        OUTDATA(23 downto 16) <= PUSHED(16 downto 9);
                        OUTDATA(26) <= PUSHED(17);
                    when "01" =>
                        OUTDATA(15 downto 8) <= PUSHED(16 downto 9);
                        OUTDATA(25) <= PUSHED(17);
                    when "10" =>
                        OUTDATA(7 downto 0) <= PUSHED(16 downto 9);
                        OUTDATA(24) <= PUSHED(17);
                    when others =>
                        NULL;
                end case;
            else
-- first cycle (repeat)
                case INDEX is
                    when "00" =>
                        OUTDATA(23 downto 16) <= PUSHED(7 downto 0);
                        OUTDATA(26) <= PUSHED(8);
                    when "01" =>
                        OUTDATA(15 downto 8) <= PUSHED(7 downto 0);
                        OUTDATA(25) <= PUSHED(8);
                    when "10" =>
                        OUTDATA(7 downto 0) <= PUSHED(7 downto 0);
                        OUTDATA(24) <= PUSHED(8);
                    when others =>
                        NULL;
```

```
            end case;
        end if;

-- update index pointer and push data if on last one
        case INDEX is
            when "00" =>
                INDEX <= "01";
                PUSH <= '0';
            when "01" =>
                INDEX <= "10";
                PUSH <= '0';
            when "10" =>
                INDEX <= "00";
                PUSH <= '1';
            when others =>
                INDEX <= "00";
                PUSH <= '0';
        end case;

    else

-- do nothing to FIFO
        PUSH <= '0';

    end if;

  end process camera_proc4;

end behaviour;
```

D.2.4.4 deltatracker.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(CLK: in STD_LOGIC;
  RESET: in STD_LOGIC;
  AD: in UNSIGNED(7 downto 0);
  BD: in UNSIGNED(7 downto 0);
  LVAL: in STD_LOGIC;
  PVAL: in STD_LOGIC;
  BITRD: in STD_LOGIC_VECTOR(15 downto 0);
  BITRA: out STD_LOGIC_VECTOR(3 downto 0);
  VAR0: in UNSIGNED(7 downto 0);
  VAR1: in UNSIGNED(7 downto 0);
  VAR2: in UNSIGNED(7 downto 0);
  VAR3: in UNSIGNED(7 downto 0);
  VAR4: in UNSIGNED(7 downto 0);
  VAR5: in UNSIGNED(7 downto 0);
  VAR6: in UNSIGNED(7 downto 0);
  VAR7: in UNSIGNED(7 downto 0);
  OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
  PUSH: out STD_LOGIC;
  CAPTUREON: in STD_LOGIC;
```

```
   OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal FIRSTPIXEL: STD_LOGIC;
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal TOBESYNCED: STD_LOGIC;
signal CURRENT: UNSIGNED(7 downto 0);
signal DATA: STD_LOGIC_VECTOR(7 downto 0);
signal PUSHED: STD_LOGIC_VECTOR(17 downto 0);
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);
signal COUNT: STD_LOGIC_VECTOR(7 downto 0);
signal CPUSH: STD_LOGIC;
signal CCOMP: STD_LOGIC;
signal CEND: STD_LOGIC;
signal CLCOMP: STD_LOGIC;
signal CPROC: STD_LOGIC;
signal CCONT: STD_LOGIC;
signal WIDE: STD_LOGIC;
signal COUNTER: UNSIGNED(10 downto 0);

signal BACKVAL: UNSIGNED(7 downto 0);
signal BACKHCOMP: UNSIGNED(7 downto 0);
signal LOWC: UNSIGNED(3 downto 0);
signal LOWCBW: UNSIGNED(3 downto 0);
signal BACKHPROC: STD_LOGIC;
signal THRES: UNSIGNED(4 downto 0);

signal TEMP0: UNSIGNED(10 downto 0);
signal TEMP1: UNSIGNED(10 downto 0);
begin

-- convert programmable parameters into something we can use
    TEMP0 <= VAR2(2 downto 0) & VAR0;
    TEMP1 <= VAR2(5 downto 3) & VAR1;
    THRES <= VAR3(4 downto 0);
    LOWCBW <= VAR4(3 downto 0);

-- stabalize camera inputs
    camera_proc1: process begin
        wait until CLK'event and CLK='0';

        CURRENT <= AD;
        LVALSIG <= LVAL;
        PVALSIG <= PVAL;

    end process camera_proc1;

    camera_proc2: process begin
        wait until CLK'event and CLK='0';

-- when pixel is valid
        if (PVALSIG='0' and CAPTUREON='1') then

-- when line is valid
           if (LVALSIG='1') then
```

```
            TOBESYNCED <= '1';

-- initialize variable on first pixel
            if ((FIRSTPIXEL='1' and OKTOPUSH='0') or PROCLINE='0') then

                FIRSTPIXEL <= '0';
                PROCLINE <= '0';
                CPROC <= '0';
                CCOMP <= '1';
                CEND <= '0';
                BACKHPROC <= '0';

            else

                PROCLINE <= '1';
                CEND <= '0';
                BACKVAL <= CURRENT;

-- increment pixel counter
                COUNTER <= COUNTER + 1;

                if ( COUNTER >= TEMP0 and COUNTER <= TEMP1 ) then

                    FIRSTPIXEL <= '0';
                    CPROC <= '1';

-- check for range
                    if ( (CURRENT >= (BACKHCOMP+THRES)) or (CURRENT <=
(BACKHCOMP-THRES)) ) then

-- defect pixel
                        DATA <= CONV_STD_LOGIC_VECTOR(CURRENT,8);
                        CCOMP <= '0';
                        BACKHPROC <= '0';

                    else

-- non-defect pixel
                        CCOMP <= '1';
                        BACKHPROC <= '1';

                    end if;

                else

-- nothing, not in range
                    CPROC <= '0';
                    BACKHPROC <= '0';

                end if;

            end if;

        else

            CCOMP <= '0';
            CEND <= '1';
```

```
-- process end of line if necessary
              if ( TOBESYNCED='1' and PROCLINE='1' ) then
                 CPROC <= '1';
              else
                 CPROC <= '0';
              end if;

-- get ready for next line
              TOBESYNCED <= '0';
              FIRSTPIXEL <= '1';
              PROCLINE <= '1';
              COUNTER <= ( others => '0' );
              BACKHPROC <= '0';

          end if;
       else

          CPROC <= '0';
          CCOMP <= '1';
          CEND <= '0';
          BACKHPROC <= '0';

       end if;

    end process camera_proc2;

-- Perform deltatracker calculations
    camera_backproc: process begin

       wait until CLK'event and CLK='0';

       if (BACKHPROC = '1') then
          if (BACKVAL > BACKHCOMP) then
             if (LOWC = LOWCBW) then
                LOWC <= (others => '0');
                if (not (BACKHCOMP = 255)) then
-- Increment background
                    BACKHCOMP <= BACKHCOMP + 1;
                end if;
             else
                LOWC <= LOWC + 1;
             end if;
          elsif (BACKVAL < BACKHCOMP) then
             if (LOWC = 0) then
                LOWC <= LOWCBW;
                if (not (BACKHCOMP = 0)) then
-- Decrement background
                    BACKHCOMP <= BACKHCOMP - 1;
                end if;
             else
                LOWC <= LOWC - 1;
             end if;
          end if;
       elsif (FIRSTPIXEL='1') then
          LOWC <= (others => '0');
          BACKHCOMP <= BACKVAL;
       end if;

    end process;
```

```
-- Perform simple RLE compression
    camera_proc3: process
    variable CASEVAR: STD_LOGIC_VECTOR(2 downto 0);
    begin

        wait until CLK'event and CLK='0';

        CASEVAR := CCOMP & CLCOMP & CEND;

-- do it on data to be processed
        if (CPROC='1') then

            case CASEVAR is

                when "011" | "111" => -- end of line, send sync and last repeat

                    PUSHED <= '1' & "00000000" & '1' & COUNT;
                    WIDE <= '1';
                    CPUSH <= '1';
                    CLCOMP <= '0';

                when "001" | "101" => -- end of line, send sync

                    PUSHED(8 downto 0) <= '1' & "00000000";
                    WIDE <= '0';
                    CPUSH <= '1';
                    CLCOMP <= '0';

                when "110" => -- repeat data, increment counter, dump if overflow

                    if (COUNT="11111111") then
                        PUSHED(8 downto 0) <= '1' & COUNT;
                        WIDE <= '0';
                        CPUSH <= '1';
                        CLCOMP <= '1';
                        COUNT <= "00000001";
                    else
                        COUNT <= COUNT + 1;
                        WIDE <= '0';
                        CPUSH <= '0';
                        CLCOMP <= '1';
                    end if;

                when "100" => -- start of RLE encoding

                    COUNT <= "00000001";
                    WIDE <= '0';
                    CPUSH <= '0';
                    CLCOMP <= '1';

                when "010" => -- send luminance and repeat of last value

                    PUSHED <= '0' & DATA & '1' & COUNT;
                    WIDE <= '1';
                    CPUSH <= '1';
                    CLCOMP <= '0';

                when "000" => -- send luminance only
```

```
                PUSHED(8 downto 0) <= '0' & DATA;
                WIDE <= '0';
                CPUSH <= '1';
                CLCOMP <= '0';

            when others => -- nothing

                NULL;

        end case;

    else

        CPUSH <= '0';

    end if;

  end process camera_proc3;

-- Dump RLE block to FIFO
  camera_proc4: process begin

    wait until CLK'event and CLK='0';

-- Push 16 bit values in 2 cycles
    if (CPUSH='1' or CCONT='1') then

        if (CPUSH='1' and WIDE='1') then
            CCONT <= '1';
        else
            CCONT <= '0';
        end if;

        if (CCONT = '1') then
-- second cycle (luminance or sync)
            case INDEX is
                when "00" =>
                    OUTDATA(23 downto 16) <= PUSHED(16 downto 9);
                    OUTDATA(26) <= PUSHED(17);
                when "01" =>
                    OUTDATA(15 downto 8) <= PUSHED(16 downto 9);
                    OUTDATA(25) <= PUSHED(17);
                when "10" =>
                    OUTDATA(7 downto 0) <= PUSHED(16 downto 9);
                    OUTDATA(24) <= PUSHED(17);
                when others =>
                    NULL;
            end case;
        else
-- first cycle (repeat)
            case INDEX is
                when "00" =>
                    OUTDATA(23 downto 16) <= PUSHED(7 downto 0);
                    OUTDATA(26) <= PUSHED(8);
                when "01" =>
                    OUTDATA(15 downto 8) <= PUSHED(7 downto 0);
                    OUTDATA(25) <= PUSHED(8);
                when "10" =>
```

```
                    OUTDATA(7 downto 0) <= PUSHED(7 downto 0);
                    OUTDATA(24) <= PUSHED(8);
                when others =>
                    NULL;
            end case;
        end if;

-- update index pointer and push data if on last one
        case INDEX is
            when "00" =>
                INDEX <= "01";
                PUSH <= '0';
            when "01" =>
                INDEX <= "10";
                PUSH <= '0';
            when "10" =>
                INDEX <= "00";
                PUSH <= '1';
            when others =>
                INDEX <= "00";
                PUSH <= '0';
        end case;

      else

-- do nothing to FIFO
        PUSH <= '0';

      end if;

   end process camera_proc4;

end behaviour;
```

D.2.4.5 fuzzy.vhd

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- for simulation
package ROMS is
  -- declare a 16x16 ROM called ROM
  constant ROM_WIDTH: INTEGER := 16;
  subtype ROM_WORD is STD_LOGIC_VECTOR (1 to ROM_WIDTH);
  subtype ROM_RANGE is INTEGER range 0 to 15;
  type ROM_TABLE is array (0 to 15) of ROM_WORD;
  constant ROM: ROM_TABLE := ROM_TABLE'(
      ROM_WORD'("1010110101101011"),              -- ROM contents
      ROM_WORD'("1010111000010101"),
      ROM_WORD'("1111110101101011"),
      ROM_WORD'("1010111111110101"),
      ROM_WORD'("1000010101101011"),
      ROM_WORD'("1010110101101011"),
      ROM_WORD'("1111110101101011"),
      ROM_WORD'("1111110101101011"),
      ROM_WORD'("1010110101101011"),
```

```
      ROM_WORD'("1000010101101011"),
      ROM_WORD'("1111110101101011"),
      ROM_WORD'("1111110101101011"),
      ROM_WORD'("1000010101101011"),
      ROM_WORD'("1010110101101011"),
      ROM_WORD'("1111110101101011"),
      ROM_WORD'("1111110101101011"));
end ROMS;

use work.ROMS.all;   -- Entity that uses ROM

-- Include necessary libraries (again)
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(
    CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    FVAL: in STD_LOGIC;
    BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal FVALSIG: STD_LOGIC;
signal TOBESYNCED: STD_LOGIC;
signal CURRENT: UNSIGNED(7 downto 0);
signal FIRSTCURRENT: STD_LOGIC;

signal COUNTER: UNSIGNED(10 downto 0);
signal TEMP0: UNSIGNED(10 downto 0);
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);

-- AVELINECURRENT is the average of gray levels in a line scanned by the camera
-- SUMLINEPIEXL is the summation of 1024 pixels in a line.

signal DIFFERENCE:SIGNED(8 downto 0);
```

```
signal PRECURRENT:SIGNED(8 downto 0) :="001111111";
signal SUMLINECURRENT: UNSIGNED(17 downto 0) :="000000000000000000";
signal THRESHOLD: UNSIGNED(4 downto 0) :="11001";
signal NUMJUMPS: UNSIGNED(3 downto 0) :="0000";
signal JUMPMOD: UNSIGNED(3 downto 0) :="0000";
signal JUMPFINAL: UNSIGNED(3 downto 0) :="0000";
signal OUTROM: STD_LOGIC_VECTOR (15 downto 0);
signal FLAG: STD_LOGIC;
signal PHASE: STD_LOGIC_VECTOR(2 downto 0);
signal AC_AVELINECURRENT: UNSIGNED(7 downto 0);
signal AVELINECURRENT: UNSIGNED(7 downto 0);
signal INCJUMP: STD_LOGIC;
constant WINDOWSIZE: INTEGER := 63;
constant WINDOWSIZE2: INTEGER := 32;
signal WINDOWOFFSET: SIGNED(8 downto 0);

-- for simulation add:
-- signalBITRD:  STD_LOGIC_VECTOR(15 downto 0);
-- signalBITRA:  STD_LOGIC_VECTOR(3 downto 0);
-- signal CLK: STD_LOGIC :='0';


begin

-- for simulation add:
-- process (CLK)
--     begin
--     if (CLK='0') then
--        CLK <= '1' after 50 ns;
--     else
--        CLK <= '0' after 50 ns;
--     end if;
-- end process;

-- BITRD <= ROM( CONV_INTEGER(BITRA));

-- convert programmable parameters into something we can use
   TEMP0 <= "0" & VAR1(1 downto 0) & VAR0;
   THRESHOLD <= VAR2(4 downto 0);
   WINDOWOFFSET <= CONV_SIGNED(VAR3,9);
   JUMPMOD <= VAR4(3 downto 0);

-- stabalize camera inputs
   camera_proc1: process begin
      wait until CLK'event and CLK='0';

      CURRENT <= AD;
      LVALSIG <= LVAL;
      PVALSIG <= PVAL;

   end process camera_proc1;

   camera_proc3: process begin

      wait until CLK'event and CLK='0';

-- calculate number of jumps from previous comparison
      if ( (DIFFERENCE >= THRESHOLD) and (NUMJUMPS < 15 ) and INCJUMP = '1')
then
```

```
            NUMJUMPS <= NUMJUMPS + 1;
        end if;

-- when pixel is valid
        if (PVALSIG='0' and CAPTUREON='1') then

-- set sync line and defect line flag
            OUTDATA(24) <= '0';
            OUTDATA(25) <= FLAG;
            OUTDATA(26) <= TOBESYNCED;

-- when line is valid
            if (LVALSIG='1') then
                PHASE <= "000";

-- initialize variable on first pixel
                if ((FIRSTCURRENT='1' and OKTOPUSH='0') or PROCLINE='0') then

                    PROCLINE <= '0';
                    INDEX <= "00";
                    PUSH <= '0';
                    TOBESYNCED <= '0';
                    FLAG <= '0';
                    FIRSTCURRENT <= '0';
                    INCJUMP <= '0';

                else

-- increment pixel counter
                    COUNTER <= COUNTER + 1;

-- check for line range
                    if ( (COUNTER >= TEMP0) and (COUNTER <= (1023 + TEMP0)) ) then

                        FIRSTCURRENT <= '0';
                        PROCLINE <= '1';

-- Calculation of the average gray level of each line begins:
                        SUMLINECURRENT <= SUMLINECURRENT+CURRENT;

                        DIFFERENCE <= ABS (PRECURRENT-CURRENT);
                        INCJUMP <= '1';

                        PRECURRENT <= CONV_SIGNED(CURRENT,9);

-- figure out which of 3 8-bit sections to write to
                        case INDEX is
                            when "00" =>
                                OUTDATA(23 downto 16) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                                INDEX <= "01";
                                PUSH <= '0';
                            when "01" =>
                                OUTDATA(15 downto 8) <=
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                                INDEX <= "10";
                                PUSH <= '0';
                            when "10" =>
                                OUTDATA(7 downto 0) <=
```

```
CONV_STD_LOGIC_VECTOR(CURRENT,8);
                        INDEX <= "00";
-- on the last one, write the 27-bit data to the FIFO
                        PUSH <= '1';
                        TOBESYNCED <= '0';
                        FLAG <= '0';
                     when others =>
                        NULL;
                  end case;

               else

                  FIRSTCURRENT <= '0';
                  PROCLINE <= '1';
                  PUSH <= '0';
                  INCJUMP <= '0';

               end if;

            end if;

         else

-- calculate algorithm stuff in phases
            case PHASE is

               when "000" =>

-- Compute average
                  AVELINECURRENT <= CONV_UNSIGNED( ABS( CONV_UNSIGNED(
SHR(SUMLINECURRENT,CONV_UNSIGNED(10,4)),8) - WINDOWOFFSET - WINDOWSIZE2 ) ,8);
                  PHASE <= "001";

-- Adjust number of jumps
                  if ( NUMJUMPS > JUMPMOD) then
                     JUMPFINAL <= NUMJUMPS - JUMPMOD;
                  else
                     JUMPFINAL <= ( others => '0' );
                  end if;

-- debug stuff
--                OUTDATA(15 downto 8) <=
CONV_STD_LOGIC_VECTOR(SUMLINECURRENT,18)(17 downto 10);
--                OUTDATA(23 downto 16) <= CONV_STD_LOGIC_VECTOR(NUMJUMPS,8);
--                PUSH <= '0';

               when "001" =>

-- Calculate RAM address
                  if (AVELINECURRENT >= WINDOWSIZE2 ) then
                     AC_AVELINECURRENT <= CONV_UNSIGNED(WINDOWSIZE2 - 1,8);
                  else
                     AC_AVELINECURRENT <= AVELINECURRENT;
                  end if;

-- Adjust number of jumps
                  if (JUMPFINAL > 7) then
                     JUMPFINAL <= CONV_UNSIGNED(7,4);
                  end if;
```

```
                    PHASE <= "101";

-- debug stuff
--                  PUSH <= '0';

                when "101" =>

-- Set RAM address lines
                    BITRA <= CONV_STD_LOGIC_VECTOR(AC_AVELINECURRENT,8)(3 downto
0);
                    PHASE <= "110";

-- debug stuff
--                  OUTDATA(7 downto 4) <=
CONV_STD_LOGIC_VECTOR(AC_AVELINECURRENT,8)(3 downto 0);
--                  PUSH <= '0';

                when "110" =>

-- Set the flag
                    FLAG <= BITRD(CONV_INTEGER(
CONV_STD_LOGIC_VECTOR(AC_AVELINECURRENT,8)(4) &
CONV_STD_LOGIC_VECTOR(JUMPFINAL,4)(2 downto 0)));
                    PHASE <= "111";
                    NUMJUMPS <= (others => '0');

-- debug stuff
--                  OUTDATA(3 downto 0) <= CONV_STD_LOGIC_VECTOR( CONV_INTEGER(
CONV_STD_LOGIC_VECTOR(AC_AVELINECURRENT,8)(4) &
CONV_STD_LOGIC_VECTOR(JUMPFINAL,4)(2 downto 0)) ,4);
--                  OUTDATA(23 downto 8) <= BITRD;
--                  PUSH <= '1';

                when others =>

                    NULL;

            end case;

-- If we didn't write all the data, do it now
            if ( (not (INDEX = "00")) and TOBESYNCED='0' ) then
                PUSH <= '1';
            else
                PUSH <= '0';
            end if;
-- get ready for next line
            INDEX <= "00";
            TOBESYNCED <= '1';
            FIRSTCURRENT <= '1';
            PROCLINE <= '1';
            COUNTER <= (others => '0');
            SUMLINECURRENT <= (others => '0'); --End of processing averaging
pixel values of each line
            INCJUMP <= '0';

        end if;
    else
-- nothing to be done, don't write
```

```
            PUSH <= '0';
            INCJUMP <= '0';
        end if;

    end process camera_proc3;

end behaviour;
```

D.2.5 Test VHDL Code

D.2.5.1 ramtest.vhd

The following code is a video processing model which pushes the RAM contents on to the FIFO. All camera luminance data is ignored.

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;
    BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal FIRSTPIXEL: STD_LOGIC;
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal PROCNEXT: STD_LOGIC;
signal COUNTER: UNSIGNED(10 downto 0);
```

```
signal ADDR: STD_LOGIC_VECTOR(3 downto 0);
begin

    camera_proc1: process begin
    wait until CLK'event and CLK='0';

-- stabalize camera inputs (simulate data rate)
    LVALSIG <= LVAL;
    PVALSIG <= PVAL;

    end process camera_proc1;

    camera_proc3: process begin
    wait until CLK'event and CLK='0';

-- when pixel is valid
    if (PVALSIG='0' and CAPTUREON='1') then

        OUTDATA(26 downto 20) <= (others => '0');

-- when line is valid
        if (LVALSIG='1') then

-- initialize variable on first pixel
           if ((FIRSTPIXEL='1' and OKTOPUSH='0') or PROCLINE='0') then

               PROCLINE <= '0';
               PUSH <= '0';
               FIRSTPIXEL <= '0';
               PROCNEXT <= '0';

           else

-- use counter as a ram address reference
                COUNTER <= COUNTER + 1;

-- set addresses for next cycle read
                BITRA <= CONV_STD_LOGIC_VECTOR(COUNTER,11)(3 downto 0);
                ADDR <= CONV_STD_LOGIC_VECTOR(COUNTER,11)(3 downto 0);
                PROCNEXT <= '1';
                if (PROCNEXT='1') then
-- store data from RAM into FIFO
                    OUTDATA(15 downto 0) <= BITRD;
                    OUTDATA(19 downto 16) <= ADDR;
                    PUSH <= '1';
                else
                    PUSH <= '0';
                end if;

                FIRSTPIXEL <= '0';
                PROCLINE <= '1';

           end if;

        else

-- no pixel, we want to simulate the same data rate too
           PUSH <= '0';
           FIRSTPIXEL <= '1';
```

```
            PROCLINE <= '1';
            COUNTER <= (others => '0');
            PROCNEXT <= '0';

        end if;
    else
-- nothing at all, no FIFO push
        PUSH <= '0';
    end if;

    end process camera_proc3;

end behaviour;
```

D.2.5.2 regtest.vhd

The following code is a video processing model which pushes the register contents on to the FIFO. All camera luminance data is ignored.

```
-- Include necessary libraries
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- define sub-system ports
entity processor is
port(CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;
    BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
    OKTOPUSH: in STD_LOGIC );
end processor;

architecture behaviour of processor is
signal FIRSTPIXEL: STD_LOGIC;
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
```

```
signal PVALSIG: STD_LOGIC;
signal TOBESYNCED: STD_LOGIC;
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);
begin

    camera_proc1: process begin
    wait until CLK'event and CLK='0';

-- stabalize camera inputs (simulate data rate)
    LVALSIG <= LVAL;
    PVALSIG <= PVAL;

    end process camera_proc1;

    camera_proc3: process begin
    wait until CLK'event and CLK='0';

-- when pixel is valid
    if (PVALSIG='0' and CAPTUREON='1') then

        OUTDATA(25 downto 24) <= (others => '0');
        OUTDATA(26) <= TOBESYNCED;

-- when line is valid
        if (LVALSIG='1') then

-- initialize variable on first pixel
           if ((FIRSTPIXEL='1' and OKTOPUSH='0') or PROCLINE='0') then

               PROCLINE <= '0';
               INDEX <= "00";
               PUSH <= '0';
               TOBESYNCED <= '0';
               FIRSTPIXEL <= '0';

           else

               FIRSTPIXEL <= '0';
               PROCLINE <= '1';

-- figure out which of 3 8-bit sections to write to
               case INDEX is
                   when "00" =>
                       OUTDATA(23 downto 16) <= CONV_STD_LOGIC_VECTOR(VAR0,8);
                       INDEX <= "01";
                       PUSH <= '0';
                   when "01" =>
                       OUTDATA(15 downto 8) <= CONV_STD_LOGIC_VECTOR(VAR1,8);
                       INDEX <= "10";
                       PUSH <= '0';
                   when "10" =>
                       OUTDATA(7 downto 0) <= CONV_STD_LOGIC_VECTOR(VAR2,8);
                       INDEX <= "00";
-- on the last one, write the 27-bit data to the FIFO
                       PUSH <= '1';
                       TOBESYNCED <= '0';
                   when others =>
                       NULL;
               end case;
```

```
            end if;

        else

-- If we didn't write all the data, do it now
            if ( (not (INDEX = "00")) and TOBESYNCED='0' ) then
                PUSH <= '1';
            else
                PUSH <= '0';
            end if;
-- get ready for next line
            INDEX <= "00";
            TOBESYNCED <= '1';
            FIRSTPIXEL <= '1';
            PROCLINE <= '1';

        end if;
    else
-- nothing to be done, don't write
        PUSH <= '0';
    end if;

    end process camera_proc3;

end behaviour;
```

D.2.5.3 fifotest.vhd

The following code is the whole complete FPGA module. No linking is required. This is used to test the FIFO push commands.

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

library SYNOPSYS;
use SYNOPSYS.ATTRIBUTES.all;

-- define system ports
entity test is
    port(CLK: in STD_LOGIC;
    FOUT: out STD_LOGIC_VECTOR(26 downto 0);
    WEN1: out STD_LOGIC;
    WEN2: out STD_LOGIC;
    WCLK: out STD_LOGIC );
end test;

architecture behaviour of test is

-- define special Xilinx XC4000 components
component MD0
```

```
   port(I: out STD_LOGIC );
end component;

component MD1
   port(O: in STD_LOGIC );
end component;

component MD2
   port(I: out STD_LOGIC );
end component;

component TCK
   port(I: out STD_LOGIC );
end component;

component TDI
   port(I: out STD_LOGIC );
end component;

component TMS
   port(I: out STD_LOGIC );
end component;

component TDO
   port(O: in STD_LOGIC );
end component;

component IBUF
   port(I: in STD_LOGIC;
   O: out STD_LOGIC );
end component;

component OBUF
   port(I: in STD_LOGIC;
   O: out STD_LOGIC );
end component;

constant CYCLE1: STD_LOGIC := '0';
constant CYCLE2: STD_LOGIC := '1';

signal tomdlpin: STD_LOGIC;
signal tomdlbuf: STD_LOGIC;
signal totdopin: STD_LOGIC;
signal totdobuf: STD_LOGIC;
signal fromtckpin: STD_LOGIC;
signal fromtckbuf: STD_LOGIC;
signal fromtdipin: STD_LOGIC;
signal fromtdibuf: STD_LOGIC;

signal CDOUT: STD_LOGIC;
signal FRS: STD_LOGIC;
signal PAF: STD_LOGIC;
signal PAE: STD_LOGIC;

signal GO: STD_LOGIC;

signal C1: STD_LOGIC_VECTOR(15 downto 0);

begin
```

```
-- Instantiate special components
   xMD1BUF: OBUF port map ( I => tomd1buf , O => tomd1pin );
   xMD1PIN: MD1 port map ( O => tomd1pin );
   tomd1buf <= CDOUT;

   xTDOBUF: OBUF port map ( I => totdobuf , O => totdopin );
   xTDOPIN: TDO port map ( O => totdopin );
   totdobuf <= FRS;

   xTCKPIN: TCK port map ( I => fromtckpin );
   xTCKBUF: IBUF port map ( I => fromtckpin , O => fromtckbuf );
   PAF <= fromtckbuf;

   xTDIPIN: TDI port map ( I => fromtdipin );
   xTDIBUF: IBUF port map ( I => fromtdipin , O => fromtdibuf );
   PAE <= fromtdibuf;

-- Set write clock
   WCLK <= CLK;

-- Let Synopsys do the state machines since we do not have any
--  asynchronous resets
   process
   begin

-- reset
       wait until CLK'event and CLK='0';

       FRS<='0';
       WEN1<='1';
       WEN2<='1';
       C1 <= (others => '0');
       FOUT <= (others => '0');

-- reset
       wait until CLK'event and CLK='0';

       FRS<='0';
       WEN1<='1';
       WEN2<='1';
       C1 <= (others => '0');
       FOUT <= (others => '0');

-- reset
       wait until CLK'event and CLK='0';

       FRS<='0';
       WEN1<='1';
       WEN2<='1';
       C1 <= (others => '0');
       FOUT <= (others => '0');

-- nothing
       wait until CLK'event and CLK='0';

       FRS<='1';
       WEN1<='1';
       WEN2<='1';
```

```
        C1 <= (others => '0');
        FOUT <= (others => '0');

-- write
        loop

            wait until CLK'event and CLK='0';

            FOUT <= (not C1(10 downto 0)) & C1;
            if (PAF='1') then
-- write away
                FRS<='1';
                WEN1<='0';
                WEN2<='1';
                C1 <= C1 + 1;
            else
-- do not write if full
                FRS<='1';
                WEN1<='1';
                WEN2<='1';
            end if;

        end loop;

    end process;

end behaviour;
```

D.3 CPLD1 Hardware Description

D.3.1 Synopsys Scripts

D.3.1.1 main.scr

```
/* read in vhdl code */
read -f vhdl main.vhd

/* Use either DSPCLK as source, or use loop through to CLKLPIN
/* create_clock -name "DSPCLK" -period 30 -waveform { "0" "15" } { "DSPCLK" } */
create_clock -name "CLKLPIN" -period 30 -waveform { "0" "15" } { "CLKLPIN" }

/* compile to target, low effect acceptable */
compile -map_effort low

set_pad_type -no_clock "*"

/* Use either DSPCLK as source, or use loop through to CLKLPIN
/* set_pad_type -clock "DSPCLK" */
set_pad_type -clock "CLKLPIN"

set_port_is_pad "*"

include mainpins.scr

insert_pads

ungroup -all -flatten

write -format xnf -hierarchy -output main.sxnf

write_script > main.dc

sh dc2ncf main.dc

sh cpld -p 9536-5VQ44 main.sxnf
```

D.3.1.2 mainpins.scr

```
set_attribute {"GSR"} "pad_location" -type string "P33"
set_attribute {"RSIN"} "pad_location" -type string "P28"
set_attribute {"RSOUT"} "pad_location" -type string "P40"

set_attribute {"DSPCLK"} "pad_location" -type string "P44"
set_attribute {"CLKLPIN"} "pad_location" -type string "P43"
set_attribute {"CLKLPOUT"} "pad_location" -type string "P42"

set_attribute {"DSPRW"} "pad_location" -type string "P36"
set_attribute {"DSPIS"} "pad_location" -type string "P34"
set_attribute {"DSPAL"} "pad_location" -type string "P39"
set_attribute {"DSPAH<0>"} "pad_location" -type string "P37"
set_attribute {"DSPAH<1>"} "pad_location" -type string "P38"

set_attribute {"TSBINT"} "pad_location" -type string "P32"
set_attribute {"TSBCA"} "pad_location" -type string "P31"
set_attribute {"FTSBCS"} "pad_location" -type string "P30"
```

```
set_attribute {"TSBCLK"} "pad_location" -type string "P29"

set_attribute {"DSPINT"} "pad_location" -type string "P12"
set_attribute {"DSPREADY"} "pad_location" -type string "P27"
set_attribute {"DSPWE"} "pad_location" -type string "P13"
set_attribute {"DSPDS"} "pad_location" -type string "P21"
set_attribute {"DSPPS"} "pad_location" -type string "P22"
set_attribute {"DSPBR"} "pad_location" -type string "P23"

set_attribute {"TRISELA"} "pad_location" -type string "P14"
set_attribute {"TRIOEA"} "pad_location" -type string "P16"
set_attribute {"TRIOEBC"} "pad_location" -type string "P18"
set_attribute {"TRICLK"} "pad_location" -type string "P19"
set_attribute {"TRICLKEN"} "pad_location" -type string "P20"

set_attribute {"SRAMOE"} "pad_location" -type string "P8"
set_attribute {"SRAMWE"} "pad_location" -type string "P7"
set_attribute {"EEPROE"} "pad_location" -type string "P5"
set_attribute {"EEPRWE"} "pad_location" -type string "P6"

set_attribute {"FIFOCLK"} "pad_location" -type string "P3"
set_attribute {"FIFOEN"} "pad_location" -type string "P2"
set_attribute {"FIFOOE"} "pad_location" -type string "P1"
```

D.3.2 VHDL Code

The 16-to-32 bit bus converter and be implemented using the SN74ABTH32316 (see Figure 3.15 on page 58) by setting LEA, LEB & LEC to low (set latches to edge triggered), SELB & SELC to high (potential input to B latch is A, and C latch is B), set $\overline{\text{CLKENA}}$ to $\overline{\text{CLKENB}}$ & $\overline{\text{CLKENC}}$ (referred to as $\overline{\text{CLKEN}}$), set CLKA to CLKB & CLKC (referred to as CLK), the DSP bus to port A, the LOW 32-bit bus to C and the HIGH 32-bit bus to B. Under a write condition, $\overline{\text{OEB}}$ and $\overline{\text{OEC}}$ are set low, $\overline{\text{OEA}}$ is set high and the DSP places the low 16-bits data on port A. The data on port A is latched (set $\overline{\text{CLKEN}}$ and CLK appropriately) where it is then driving port B. The DSP then places the high 16-bit data on port A where the data is latched again. The data from port B is latched and is driving port C and the data from port A is latched and is driving port B. At this point the low data placed on port A is now on C (LOW) and the high data places on A is now on B (HIGH). A read from the 32-bit is similar by reversing the data flow and setting $\overline{\text{OEB}}$ and $\overline{\text{OEC}}$ high and $\overline{\text{OEA}}$ low. The data on port B and C is latched, and C (LOW) is driving port A which is read by the DSP. $\overline{\text{OEB}}$ and $\overline{\text{OEC}}$ are then set low where then port B (HIGH) is then driving C. The data is latched again where data on B is moved to C which is driving A where the DSP reads it.

D.3.2.1 main.vhd

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define system ports
entity cpld_main is
    port(DSPCLK: in STD_LOGIC;
    DSPIS: in STD_LOGIC;
    DSPPS: in STD_LOGIC;
    DSPDS: in STD_LOGIC;
    DSPBR: in STD_LOGIC;
    DSPRW: in STD_LOGIC;
    DSPWE: in STD_LOGIC;
    DSPREADY: buffer STD_LOGIC;
    DSPINT: out STD_LOGIC;
    DSPAL: in STD_LOGIC;
    DSPAH: in STD_LOGIC_VECTOR(1 downto 0); -- A14 & A13

    RSIN: in STD_LOGIC;
    RSOUT: buffer STD_LOGIC;
    GSR: in STD_LOGIC;

    CLKLPIN: in STD_LOGIC;
    CLKLPOUT: out STD_LOGIC;

    TRISELA: out STD_LOGIC;

-- TRISELB = 1 (A)
-- TRISELC = 1 (B)
-- TRILEA,TRILEB,TRILEC = 0
-- TRICLKA,TRICLKB,TRICLKC = TRICLK
--  TRICLKENA,TRICLKENB,TRICLKENC = 0

    TRIOEA: out STD_LOGIC;
    TRIOEBC: out STD_LOGIC;
    TRICLK: out STD_LOGIC;
    TRICLKEN: out STD_LOGIC;

    TSBCLK: buffer STD_LOGIC;
    FTSBCS: out STD_LOGIC;
    TSBCA: in STD_LOGIC;
--  TSBRW = DSPRW
    TSBINT: in STD_LOGIC;
--  TSBRS: out STD_LOGIC;

    FIFOCLK: out STD_LOGIC;
    FIFOEN: out STD_LOGIC;
    FIFOOE: buffer STD_LOGIC;

    SRAMOE: out STD_LOGIC;
    SRAMWE: out STD_LOGIC;

    EEPROE: out STD_LOGIC;
    EEPRWE: out STD_LOGIC );

end cpld_main;
```

```
architecture behaviour of cpld_main is
constant CYCLE1: STD_LOGIC := '0';
constant CYCLE2: STD_LOGIC := '1';
signal CLK: STD_LOGIC;
signal LASTFIFO: STD_LOGIC;
signal iTSBCA: STD_LOGIC;
signal DSP_STATE: STD_LOGIC_VECTOR(2 downto 0);
signal RCOUNT: STD_LOGIC_VECTOR(8 downto 0);
signal iTSBINT: STD_LOGIC;
signal DSPINT_STATE: STD_LOGIC_VECTOR(2 downto 0);

-- signal TSBCASIG: STD_LOGIC;
begin

-- note the use of many processes
--  this is to make simulation easier

-- depends if we used the skewed clock or the direct clock
--  CLK <= DSPCLK;
    CLK <= CLKLPIN;

    set_tsbint: process(GSR,CLK)
    begin

-- under reset, make sure there is not interrupt to the DSP set
        if (GSR = '0') then
            iTSBINT <= '1';
        elsif (CLK'event and CLK='1') then
            iTSBINT <= TSBINT;
        end if;

    end process;

    dsp_manage_int: process(GSR,CLK)
    begin

-- under reset, make sure there is no interrupt to the DSP set
        if (GSR = '0') then
            DSPINT <= '1';
            DSPINT_STATE <= "000";
        elsif (CLK'event and CLK='0') then

-- when an interrupr occurs, the DSP's INT line must be pulsed
--  for continuous interrupts
            case DSPINT_STATE is
                when "000" =>

-- interrupt, set low (stage 1)
                    if (iTSBINT = '0') then
                        DSPINT <= iTSBINT;
                        DSPINT_STATE <= "001";
                    else
-- no interrupt
                        DSPINT <= '1';
                        DSPINT_STATE <= "000";
                    end if;

                when "001" =>
```

```
-- interrupt, set low (stage 2)
                DSPINT <= iTSBINT;
                DSPINT_STATE <= "010";

            when "010" =>

-- interrupt, set low (stage 3)
                DSPINT <= iTSBINT;
                DSPINT_STATE <= "011";

            when "011" =>

-- interrupt, set low (stage 4)
                DSPINT <= iTSBINT;
                DSPINT_STATE <= "100";

            when "100" =>

-- interrupt, set low (stage 5)
                DSPINT <= iTSBINT;
                DSPINT_STATE <= "101";

            when "101" =>

-- interrupt, set high (stage 6)
                DSPINT <= '1';
                DSPINT_STATE <= "110";

            when "110" =>

-- interrupt, set high (stage 7)
                DSPINT <= '1';
                DSPINT_STATE <= "111";

            when "111" =>

-- interrupt, set high (stage 8)
                DSPINT <= '1';
                DSPINT_STATE <= "000";

            when others =>

                NULL;

        end case;

    end if;

end process;

-- set SRAM control lines
    set_sram_oe_we: process(DSPDS,DSPPS,DSPRW,DSPBR,DSPWE,GSR)
    begin

        if (GSR = '0') then
-- reset, turn everything off
            SRAMOE <= '1';
            SRAMWE <= '1';
```

```
       elsif ((DSPDS='0' or DSPPS='0') and DSPBR='1') then
-- external data and program access
           SRAMOE <= not DSPRW;
           SRAMWE <= DSPWE;
       else
-- other condition, turn it off
           SRAMOE <= '1';
           SRAMWE <= '1';
       end if;

   end process;

-- set EEPROM control lines
   set_eepr_oe_we: process(DSPDS,DSPRW,DSPBR,DSPWE,GSR)
   begin

   if (GSR = '0') then
-- reset, turn everything off
       EEPROE <= '1';
       EEPRWE <= '1';
   elsif (DSPDS='0' and DSPBR='0') then
-- external data access
       EEPROE <= not DSPRW;
       EEPRWE <= DSPWE;
   else
-- other condition, turn it off
       EEPROE <= '1';
       EEPRWE <= '1';
   end if;

   end process;

-- stabilize TSBCA line
   tsbca_stable: process(GSR,CLK)
   begin

   if (GSR = '0') then
       iTSBCA <= '1';
   elsif (CLK'event and CLK='1') then
       iTSBCA <= TSBCA;
   end if;

   end process;

-- manage clock bus operations
   dsp_bus_manage: process(GSR,DSP_STATE)
   begin

-- reset, initialize variables
   if (GSR = '0') then
       LASTFIFO <= '0';
       DSPREADY <= '1';
       FIFOOE <= '1';
       TRIOEBC <= '1';
       TRICLKEN <= '1';
       FIFOEN <= '1';
       FTSBCS <= '1';

       DSP_STATE <= "000";
```

```
   elsif (CLK'event and CLK='0') then

      case DSP_STATE is

         when "000" =>

            if (DSPIS = '0') then

-- check for IO read/write (static lines)
               if (DSPAH = "01") then

                  LASTFIFO <= LASTFIFO;
                  DSPREADY <= '1';
                  FIFOOE <= '1';
                  TRIOEBC <= '1';
                  TRICLKEN <= '1';
                  FIFOEN <= '1';
                  FTSBCS <= '1';

-- write requires extra cycle
                  if (DSPRW='0') then
                     DSP_STATE <= "001";
                  else
                     DSP_STATE <= "000";
                  end if;

-- FIFO read
               elsif (DSPAH = "10" and DSPRW = '1') then

-- if last read operation was not a fifo read, enable fifo outputs
                  if (LASTFIFO = '0') then
-- Do operation two cycle
                     LASTFIFO <= '1';
                     DSPREADY <= '0';
                     FIFOOE <= '0';
                     TRIOEBC <= '1';
                     TRICLKEN <= '1';
                     FIFOEN <= '1';
                     FTSBCS <= '1';

                     DSP_STATE <= "010";
                  else
-- Do operation single cycle
                     LASTFIFO <= '1';
                     DSPREADY <= '1';
                     FIFOOE <= '0';
                     TRIOEBC <= '1';
                     TRICLKEN <= '0';
                     FIFOEN <= '0';
                     FTSBCS <= '1';

                     DSP_STATE <= "000";
                  end if;

-- TSB Operation
               elsif (DSPAH="11") then

-- if last read operation was a fifo, disable fifo outputs
```

```
                    LASTFIFO <= '0';
                    FIFOOE <= '1';
                    FIFOEN <= '1';

                    if (DSPRW='1') then
-- Read
                    -- TSBRW = DSPRW outside

                        TRIOEBC <= '1';

                        if (DSPAL='0') then

-- Low word, start read opeation, multi cycle
                            DSPREADY <= '0';
                            TRICLKEN <= '0';
                            FTSBCS <= '0';

                            DSP_STATE <= "011";

                        else
-- High word, dump bus, single cycle

                            DSPREADY <= '1';
                            TRICLKEN <= '1';
                            FTSBCS <= '1';

                            DSP_STATE <= "000";

                        end if;

                    else

-- Write
                        TRIOEBC <= '0';

                        if (DSPAL='0') then

-- Low word, dump bus, two cycle
                            DSPREADY <= '1';
                            TRICLKEN <= '0';
                            FTSBCS <= '1';

                            DSP_STATE <= "100";

                        else
-- High word, start write opeation, multi cycle

                            DSPREADY <= '0';
                            TRICLKEN <= '0';
                            FTSBCS <= '1';

                            DSP_STATE <= "101";

                        end if;

                    end if;

                else
```

```
-- Read or Write to unknown area, do nothing
                    DSPREADY <= '1';
                    LASTFIFO <= LASTFIFO;
                    FIFOOE <= '1';
                    TRIOEBC <= '1';
                    TRICLKEN <= '1';
                    FIFOEN <= '1';
                    FTSBCS <= '1';

                    DSP_STATE <= "000";

                end if;

            else

-- Do nothing
                DSPREADY <= '1';
                LASTFIFO <= LASTFIFO;
                FIFOOE <= FIFOOE;
                TRIOEBC <= '1';
                TRICLKEN <= '1';
                FIFOEN <= '1';
                FTSBCS <= '1';

                DSP_STATE <= "000";

            end if;

        when "001" =>

-- I/O write cycle 2
            LASTFIFO <= LASTFIFO;
            DSPREADY <= '1';
            FIFOOE <= '1';
            TRIOEBC <= '1';
            TRICLKEN <= '1';
            FIFOEN <= '1';
            FTSBCS <= '1';

            DSP_STATE <= "000";

        when "010" =>

-- Read FIFO cycle 2
            LASTFIFO <= '1';
            DSPREADY <= '1';
            FIFOOE <= '0';
            TRIOEBC <= '1';
            TRICLKEN <= '0';
            FIFOEN <= '0';
            FTSBCS <= '1';

            DSP_STATE <= "000";

        when "011" =>
-- TSB Read cycle 2
            LASTFIFO <= '0';
            FIFOOE <= '1';
            FIFOEN <= '1';
```

```
                TRIOEBC <= '1';

--              FTSBCS <= '1';

                if (iTSBCA = '0') then

                    FTSBCS <= '1';
                    DSPREADY <= '1';
                    TRICLKEN <= '1';

                    DSP_STATE <= "000";

                else

                    FTSBCS <= '0';
                    DSPREADY <= '0';
                    TRICLKEN <= '0';

                    DSP_STATE <= "011";

                end if;

            when "100" =>

-- TSB Write cycle 2
                LASTFIFO <= '0';
                FIFOOE <= '1';
                FIFOEN <= '1';
                TRIOEBC <= '0';

                DSPREADY <= '1';
                TRICLKEN <= '1';
                FTSBCS <= '1';

                DSP_STATE <= "000";

            when "101" =>

-- TSB write cycle 2
                LASTFIFO <= '0';
                FIFOOE <= '1';
                FIFOEN <= '1';
                TRIOEBC <= '0';

                DSPREADY <= '0';
                TRICLKEN <= '1';
                FTSBCS <= '0';

                DSP_STATE <= "111";

            when "111" =>

-- Wait for end of TBS cycle
                LASTFIFO <= '0';
                FIFOOE <= '1';
                FIFOEN <= '1';
                TRIOEBC <= '0';

                TRICLKEN <= '1';
```

```
--              FTSBCS <= '1';

                if (iTSBCA = '0') then

                    FTSBCS <= '1';
                    DSPREADY <= '1';

                    DSP_STATE <= "100";

                else

                    FTSBCS <= '0';
                    DSPREADY <= '0';

                    DSP_STATE <= "111";

                end if;

            when others =>

                NULL;

        end case;

    end if;

    end process;

-- Adjust value for bus exchanger
    TRISELA <= not DSPAL;
    TSBCLK <= CLK;
    TRICLK <= CLK;
    FIFOCLK <= CLK;

    CLKLPOUT <= DSPCLK;

    set_trioea: process(DSPAH,DSPIS,DSPRW,GSR)
    begin

-- Set bus exchange lines based on DSP input
        if (GSR = '0') then
            TRIOEA <= '1';
        elsif (DSPAH(1)='1' and DSPIS='0') then
            TRIOEA <= not DSPRW;
        else
            TRIOEA <= '1';
        end if;

    end process;

--  RSOUT <= not RSIN;

-- Watch dog
    set_reset: process(RSIN,CLK)
    begin

-- Reset watch dog on reset
        if (RSIN='1') then
            RCOUNT <= ( others => '0' );
```

```
            RSOUT <= '0';
        elsif (CLK'event and CLK='0') then
-- If not under reset, and DSP ready is ok, don't reset
            if (RSOUT='1' and DSPREADY='1') then
                RCOUNT <= ( others => '0' );
            else
-- If DSP ready is low for too long, reset system
                RCOUNT <= RCOUNT + 1;
                if (RCOUNT= "111111111" ) then
                    if (RSOUT = '1') then
                        RSOUT <= '0';
                    else
                        RSOUT <= '1';
                    end if;
                end if;
            end if;

    end process;

end behaviour;
```

D.4 CPLD2 Hardware Description

D.4.1 Synopsys Scripts

D.4.1.1 second.scr

```
/* read in vhdl code */
read -f vhdl second.vhd

create_clock -name "DSPCLK" -period 30 -waveform { "0" "15" } { "DSPCLK" }

/* compile to target, low effect acceptable */
compile -map_effort low

set_pad_type -no_clock "*"
set_pad_type -clock "DSPCLK"
set_port_is_pad "*"

include secondpins.scr

insert_pads

ungroup -all -flatten

write -format xnf -hierarchy -output second.sxnf

write_script > second.dc

sh dc2ncf second.dc

sh cpld -p 9536-5VQ44 second.sxnf
```

D.4.1.2 secondpins.scr

```
set_attribute {"GSR"} "pad_location" -type string "P33"
set_attribute {"DSPCLK"} "pad_location" -type string "P43"
set_attribute {"DSPRW"} "pad_location" -type string "P41"
set_attribute {"DSPIS"} "pad_location" -type string "P42"
set_attribute {"DSPXF"} "pad_location" -type string "P3"
set_attribute {"DSPBIO"} "pad_location" -type string "P1"
set_attribute {"DSPAL<0>"} "pad_location" -type string "P36"
set_attribute {"DSPAL<1>"} "pad_location" -type string "P37"
set_attribute {"DSPAL<2>"} "pad_location" -type string "P38"
set_attribute {"DSPAH<0>"} "pad_location" -type string "P40"
set_attribute {"DSPAH<1>"} "pad_location" -type string "P39"

set_attribute {"DSPCLKRX"} "pad_location" -type string "P14"
set_attribute {"DSPDX"} "pad_location" -type string "P22"
set_attribute {"DSPFSX"} "pad_location" -type string "P23"
set_attribute {"DSPDR"} "pad_location" -type string "P12"
set_attribute {"DSPFSR"} "pad_location" -type string "P13"

set_attribute {"DSPTOUT"} "pad_location" -type string "P44"

set_attribute {"TSBRS"} "pad_location" -type string "P2"

set_attribute {"DALSARX"} "pad_location" -type string "P19"
```

```
set_attribute {"DALSATX"} "pad_location" -type string "P18"
set_attribute {"DALSACTS"} "pad_location" -type string "P16"
set_attribute {"DALSARTS"} "pad_location" -type string "P20"

set_attribute {"FCCLK"} "pad_location" -type string "P8"
set_attribute {"FDIN"} "pad_location" -type string "P7"
set_attribute {"FPROG"} "pad_location" -type string "P6"
set_attribute {"FDONE"} "pad_location" -type string "P5"
set_attribute {"FINIT"} "pad_location" -type string "P34"
set_attribute {"FCDOUT"} "pad_location" -type string "P31"
set_attribute {"FSPARE"} "pad_location" -type string "P29"

set_attribute {"PAF"} "pad_location" -type string "P28"
set_attribute {"PAE"} "pad_location" -type string "P27"

set_attribute {"EXSYNC"} "pad_location" -type string "P30"
set_attribute {"SPARE1"} "pad_location" -type string "P32"
set_attribute {"TSBGRFEMP"} "pad_location" -type string "P21"
```

D.4.2 VHDL Code

D.4.2.1 second.vhd

```
-- Include necessary libraries
library IEEE,GTECH;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use GTECH.GTECH_components.all;

-- define system ports
entity cpld_second is
    port(DSPCLK: in STD_LOGIC;
    DSPIS: in STD_LOGIC;
    DSPRW: in STD_LOGIC;
    DSPXF: in STD_LOGIC;
    DSPBIO: out STD_LOGIC;
    DSPAL: in STD_LOGIC_VECTOR(2 downto 0);
    DSPAH: in STD_LOGIC_VECTOR(1 downto 0); -- A14 & A13

    DSPCLKRX: buffer STD_LOGIC;
    DSPDX: in STD_LOGIC;
    DSPFSX: in STD_LOGIC;
    DSPDR: out STD_LOGIC;
    DSPFSR: out STD_LOGIC;

    DALSARX: in STD_LOGIC;
    DALSATX: out STD_LOGIC;

    DSPTOUT: in STD_LOGIC;

    TSBRS: out STD_LOGIC;
    TSBGRFEMP: in STD_LOGIC;

    FCCLK: out STD_LOGIC;
    FDIN: out STD_LOGIC;
    FPROG: out STD_LOGIC;
    FSPARE: out STD_LOGIC;
```

```
    FDONE: in STD_LOGIC;
    FINIT: in STD_LOGIC;
    FCDOUT: in STD_LOGIC;

    PAF: in STD_LOGIC;
    PAE: in STD_LOGIC;

    EXSYNC: in STD_LOGIC; -- out later
    SPARE1: in STD_LOGIC; -- out later

    DALSACTS: in STD_LOGIC;
    DALSARTS: out STD_LOGIC;

    GSR: in STD_LOGIC );

end cpld_second;

-- Static Inputs:
-- FPGA: INIT, DONE
--     FIFO: PAE, PAF
--     TSB: -
--     DALSA: CTS
-- Static Outputs:
-- FPGA: DIN,CCLK,PROG
--     FIFO: -
--     TSB: RS
--     DALSA: RTS

architecture behaviour of cpld_second is
signal INTRCOUNT: STD_LOGIC_VECTOR(2 downto 0);
signal INTXCOUNT: STD_LOGIC_VECTOR(2 downto 0);
signal RCOUNT: STD_LOGIC_VECTOR(2 downto 0);
signal TCOUNT: STD_LOGIC_VECTOR(2 downto 0);
signal iDALSARX: STD_LOGIC;
signal DSP_STATE: STD_LOGIC;
signal RX_STATE: STD_LOGIC_VECTOR(3 downto 0);
signal TX_STATE: STD_LOGIC_VECTOR(3 downto 0);
signal STATICOUTP: STD_LOGIC_VECTOR(7 downto 0);
signal STATICINP: STD_LOGIC_VECTOR(7 downto 0);
begin

-- Set spare lines to high-impedance
-- FSPARE <= 'Z';
-- EXSYNC <= 'Z';
-- SPARE1 <= 'Z';

    static_outputs: process(GSR,STATICOUTP)
    begin

-- During reset, set some lines
    if (GSR = '0') then

        TSBRS <= '0';

        FCCLK <= '0';
        FDIN <= '0';
        FPROG <= '0';

        DALSARTS <= '0';
```

```
    else

-- Not in reset set static IOs to appropriate pins
        TSBRS <= STATICOUTP(0);

        FCCLK <= STATICOUTP(1);
        FDIN <= STATICOUTP(2);
        FPROG <= STATICOUTP(3);

        DALSARTS <= STATICOUTP(4);

    end if;

    end process;

-- Set other static IOs which states do not matter at reset
    STATICINP(0) <= FDONE;
    STATICINP(1) <= FINIT;
    STATICINP(2) <= FCDOUT;

    STATICINP(3) <= PAF;
    STATICINP(4) <= PAE;

    STATICINP(5) <= DALSACTS;

    STATICINP(6) <=TSBGRFEMP;

    dsp_bus_manage: process(GSR,DSPCLK)
    begin

-- Set some stuff during reset
    if (GSR = '0') then
        DSPBIO <= '0';
        STATICOUTP <= "00000000";
        DSP_STATE <= '0';

    elsif (DSPCLK'event and DSPCLK='1') then

-- Handle Static IO interface
        case DSP_STATE is

            when '0' =>

-- check for IO read/write
                if (DSPIS='0' and DSPAH="01") then

                    if (DSPRW='1') then

-- Read
                        DSPBIO <= STATICINP( conv_integer(DSPAL(2 downto 0)) );
                        DSP_STATE <= '0';

                    else

-- Write first cycle
                        STATICOUTP( conv_integer(DSPAL(2 downto 0)) ) <= DSPXF;
                        DSP_STATE <= '1';
```

```
                end if;

            end if;

        when '1' =>

-- Write second cycle
            STATICOUTP( conv_integer(DSPAL(2 downto 0)) ) <= DSPXF;
            DSP_STATE <= '0';

        when others =>

            NULL;

    end case;

end if;

end process;

-- Set DSP synchronous serial transmit/receive clock
DSPCLKRX <= TCOUNT(2);

-- Synchronous to asynchronous conversion
dsp_toutr: process(GSR,DSPTOUT)
begin

-- During reset, stop counters
if (GSR = '0') then

    TCOUNT <= (others => '0');
    RCOUNT <= (others => '0');
    DALSATX <= '1';
    TX_STATE <= (others => '0');
    iDALSARX <= '1';

-- For DSP provided clock
elsif (DSPTOUT'event and DSPTOUT='1') then

-- Increment tranmission counter/clock
    TCOUNT <= TCOUNT + 1;

-- Stabalize signal
    iDALSARX <= DALSARX;
-- Check to see if it changed
    if ( ( DALSARX = (not iDALSARX) ) and DALSARX='0' and RX_STATE="0000")
then
-- Manage receiver clock
        RCOUNT <= (others => '0');
    else
        RCOUNT <= RCOUNT + 1;
    end if;

    if (TCOUNT="000") then

        case TX_STATE is

            when "0000" =>
```

```
            if (DSPFSX='1') then
                DALSATX <= '0'; -- start bit
                TX_STATE <= "0001";
            else
                DALSATX <= '1'; -- nothing
                TX_STATE <= "0000";
            end if;

        when "0001" => -- bit 7

            DALSATX <= DSPDX;
            TX_STATE <= "0010";

        when "0010" => -- bit 6

            DALSATX <= DSPDX;
            TX_STATE <= "0011";

        when "0011" => -- bit 5

            DALSATX <= DSPDX;
            TX_STATE <= "0100";

        when "0100" => -- bit 4

            DALSATX <= DSPDX;
            TX_STATE <= "0101";

        when "0101" => -- bit 3

            DALSATX <= DSPDX;
            TX_STATE <= "0110";

        when "0110" => -- bit 2

            DALSATX <= DSPDX;
            TX_STATE <= "0111";

        when "0111" => -- bit 1

            DALSATX <= DSPDX;
            TX_STATE <= "1000";

        when "1000" => -- bit 0

            DALSATX <= DSPDX;
            TX_STATE <= "1001";

        when "1001" => -- stop bit

            DALSATX <= '1';
            TX_STATE <= "0000";

        when others => -- others is nothing

            DALSATX <= '1';
            TX_STATE <= "0000";

    end case;
```

```
      end if;

   end if;

   end process;

-- Asynchronous to Synchronous conversion
   dsp_toutf: process(GSR,DSPTOUT)
   begin

-- Setup states at reset
   if (GSR = '0') then

      RX_STATE <= (others => '0');
      DSPDR <= '0';
      DSPFSR <= '0';

-- For DSP provided clock
   elsif (DSPTOUT'event and DSPTOUT='0') then

-- Start 4 cycles after signal change
      if (RCOUNT="100") then

         case RX_STATE is

            when "0000" =>

               DSPDR <= iDALSARX;

               if (iDALSARX = '0') then -- start bit
                  DSPFSR <= '1';
                  RX_STATE <= "0001";

               else
                  DSPFSR <= '0';
                  RX_STATE <= "0000";

               end if;

            when "0001" => -- bit 7

               DSPFSR <= '0';
               DSPDR <= iDALSARX;
               RX_STATE <= "0010";

            when "0010" => -- bit 6

               DSPFSR <= '0';
               DSPDR <= iDALSARX;
               RX_STATE <= "0011";

            when "0011" => -- bit 5

               DSPFSR <= '0';
               DSPDR <= iDALSARX;
               RX_STATE <= "0100";

            when "0100" => -- bit 4
```

```
                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "0101";

            when "0101" => -- bit 3

                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "0110";

            when "0110" => -- bit 2

                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "0111";

            when "0111" => -- bit 1

                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "1000";

            when "1000" => -- bit 0

                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "1001";

            when "1001" => -- stop bit (can be checked by DSP SW)

                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "0000";

            when others =>

                DSPFSR <= '0';
                DSPDR <= iDALSARX;
                RX_STATE <= "0000";

        end case;

      end if;

   end if;

   end process;

end behaviour;
```

D.5 Main PC Host Software Code

D.5.1 Workspace file

D.5.1.1 Larch.dsw

```
Microsoft Developer Studio Workspace File, Format Version 6.00
# WARNING: DO NOT EDIT OR DELETE THIS WORKSPACE FILE!

###############################################################################

Project: "Deltatracker"=..\Deltatracker\Deltatracker.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

###############################################################################

Project: "Focus"=..\Focus\Focus.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

###############################################################################

Project: "Fuzzy"=..\Fuzzy\Fuzzy.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

###############################################################################

Project: "Larch"=.\Larch.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
    Begin Project Dependency
    Project_Dep_Name Lineup
    End Project Dependency
```

```
    Begin Project Dependency
    Project_Dep_Name Focus
    End Project Dependency
    Begin Project Dependency
    Project_Dep_Name Minmax
    End Project Dependency
    Begin Project Dependency
    Project_Dep_Name Deltatracker
    End Project Dependency
    Begin Project Dependency
    Project_Dep_Name Fuzzy
    End Project Dependency
}}}

###############################################################################

Project: "Lineup"=..\Lineup\Lineup.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

###############################################################################

Project: "Minmax"=..\Minmax\Minmax.dsp - Package Owner=<4>

Package=<5>
{{{
}}}

Package=<4>
{{{
}}}

###############################################################################

Global:

Package=<5>
{{{
}}}

Package=<3>
{{{
}}}

###############################################################################
```

D.5.2 Project file

D.5.2.1 Larch.dsp

```
# Microsoft Developer Studio Project File - Name="Larch" - Package Owner=<4>
```

```
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Application" 0x0101

CFG=Larch - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Larch.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Larch.mak" CFG="Larch - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Larch - Win32 Release" (based on "Win32 (x86) Application")
!MESSAGE "Larch - Win32 Debug" (based on "Win32 (x86) Application")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Larch - Win32 Release"

# PROP BASE Use_MFC 6
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 6
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_AFXDLL" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MD /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_AFXDLL" /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /o "NUL" /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /o "NUL" /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG" /d "_AFXDLL"
# ADD RSC /l 0x409 /d "NDEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 /nologo /subsystem:windows /machine:I386
# ADD LINK32 /nologo /subsystem:windows /machine:I386

!ELSEIF  "$(CFG)" == "Larch - Win32 Debug"
```

```
# PROP BASE Use_MFC 6
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 6
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MDd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_AFXDLL" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /G5 /MDd /W3 /Gm /GX /ZI /Od /I "..\Adaptec\include" /D "WIN32"
/D "_DEBUG" /D "_WINDOWS" /D "_AFXDLL" /FR /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /o "NUL" /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /o "NUL" /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG" /d "_AFXDLL"
# ADD RSC /l 0x409 /d "_DEBUG" /d "_AFXDLL"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 /nologo /subsystem:windows /debug /machine:I386 /pdbtype:sept
# ADD LINK32 /nologo /subsystem:windows /debug /machine:I386 /pdbtype:sept /
libpath:"..\Adaptec\lib"
# SUBTRACT LINK32 /profile /map

!ENDIF

# Begin Target

# Name "Larch - Win32 Release"
# Name "Larch - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\CamInt.cpp
# End Source File
# Begin Source File

SOURCE=.\CamParams.cpp
# End Source File
# Begin Source File

SOURCE=.\ChildFrm.cpp
# End Source File
# Begin Source File

SOURCE=.\Image.cpp
# End Source File
# Begin Source File

SOURCE=.\Imageline.cpp
# End Source File
# Begin Source File
```

```
SOURCE=.\Larch.cpp
# End Source File
# Begin Source File

SOURCE=.\Larch.rc
# End Source File
# Begin Source File

SOURCE=.\LarchCptDoc.cpp
# End Source File
# Begin Source File

SOURCE=.\LarchCptView.cpp
# End Source File
# Begin Source File

SOURCE=.\MainFrm.cpp
# End Source File
# Begin Source File

SOURCE=.\Params.cpp
# End Source File
# Begin Source File

SOURCE=.\Physical.cpp
# End Source File
# Begin Source File

SOURCE=.\StdAfx.cpp
# ADD CPP /Yc"stdafx.h"
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\CamInt.h
# End Source File
# Begin Source File

SOURCE=.\CamParams.h
# End Source File
# Begin Source File

SOURCE=.\ChildFrm.h
# End Source File
# Begin Source File

SOURCE=.\Image.h
# End Source File
# Begin Source File

SOURCE=.\Imageline.h
# End Source File
# Begin Source File

SOURCE=.\Larch.h
# End Source File
```

```
# Begin Source File

SOURCE=.\LarchCptDoc.h
# End Source File
# Begin Source File

SOURCE=.\LarchCptView.h
# End Source File
# Begin Source File

SOURCE=.\MainFrm.h
# End Source File
# Begin Source File

SOURCE=.\Params.h
# End Source File
# Begin Source File

SOURCE=.\Physical.h
# End Source File
# Begin Source File

SOURCE=.\Processor.h
# End Source File
# Begin Source File

SOURCE=.\ProcessorCode.h
# End Source File
# Begin Source File

SOURCE=.\Resource.h
# End Source File
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;cnt;rtf;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\res\Larch.ico
# End Source File
# Begin Source File

SOURCE=.\res\Larch.rc2
# End Source File
# Begin Source File

SOURCE=.\res\LarchCptDoc.ico
# End Source File
# Begin Source File

SOURCE=.\res\Toolbar.bmp
# End Source File
# End Group
# Begin Group "Library Files"
```

```
# PROP Default_Filter "lib"
# Begin Source File

SOURCE=..\Adaptec\lib\Wnpapi32.lib
# End Source File
# End Group
# Begin Source File

SOURCE=.\Larch.reg
# End Source File
# End Target
# End Project
```

D.5.3 Resources

D.5.3.1 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Larch.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_LARCHTYPE                   129
#define ID_STATUS_CAPTURE               130
#define ID_STATUS_VALUE                 131
#define IDD_PARAMS                      132
#define IDD_CAMERA                      133
#define IDC_EDIT1                       1000
#define IDC_SPIN1                       1001
#define IDC_EDITGAIN1                   1001
#define IDC_EDIT2                       1002
#define IDC_SPINGAIN1                   1002
#define IDC_SPIN2                       1003
#define IDC_EDITGAIN2                   1003
#define IDC_EDIT3                       1004
#define IDC_SPINGAIN2                   1004
#define IDC_SPIN3                       1005
#define IDC_STATICGAIN1                 1005
#define IDC_EDIT4                       1006
#define IDC_STATICGAIN2                 1006
#define IDC_SPIN4                       1007
#define IDC_EDITOFFSET1                 1007
#define IDC_EDIT5                       1008
#define IDC_SPINOFFSET1                 1008
#define IDC_SPIN5                       1009
#define IDC_STATICOFFSET1               1009
#define IDC_EDIT6                       1010
#define IDC_EDITOFFSET2                 1010
#define IDC_SPIN6                       1011
#define IDC_SPINOFFSET2                 1011
#define IDC_EDIT7                       1012
#define IDC_STATICOFFSET2               1012
#define IDC_SPIN7                       1013
#define IDC_EDIT8                       1014
#define IDC_SPIN8                       1015
#define IDC_TEXT1                       1016
#define IDC_TEXT2                       1017
```

```
#define IDC_TEXT3                       1018
#define IDC_TEXT4                       1019
#define IDC_TEXT5                       1020
#define IDC_TEXT6                       1021
#define IDC_TEXT7                       1022
#define IDC_TEXT8                       1023
#define IDC_TITLE                       1024
#define ID_CAPTURE_START                32771
#define ID_CAPTURE_STOP                 32772
#define ID_CAPTURE_SCROLL               32773
#define ID_KEY_DOWN                     32774
#define ID_KEY_UP                       32775
#define ID_KEY_LEFT                     32776
#define ID_KEY_RIGHT                    32777
#define ID_KEY_PAGEDOWN                 32778
#define ID_KEY_PAGEUP                   32779
#define ID_KEY_HOME                     32780
#define ID_KEY_END                      32781
#define ID_CAPTURE_PARAMTERS            32782
#define ID_CAMERA_LOADVIDEOPROCESSOR    32784
#define ID_CAMERA_PARAMETERS            32786

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        134
#define _APS_NEXT_COMMAND_VALUE         32788
#define _APS_NEXT_CONTROL_VALUE         1026
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.5.3.2 Larch.rc

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32
```

```
#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif\r\n"
    "#include ""res\\Larch.rc2""  // non-Microsoft Visual C++ edited
resources\r\n"
    "#include ""afxres.rc""         // Standard components\r\n"
    "#endif\0"
END

#endif    // APSTUDIO_INVOKED


//////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\Larch.ico"
IDR_LARCHTYPE           ICON    DISCARDABLE     "res\\LarchCptDoc.ico"

//////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDR_MAINFRAME           BITMAP  MOVEABLE PURE   "res\\Toolbar.bmp"

//////////////////////////////////////////////////////////////////////////////
//
// Toolbar
//
```

```
IDR_MAINFRAME TOOLBAR DISCARDABLE  16, 15
BEGIN
    BUTTON      ID_FILE_NEW
    BUTTON      ID_FILE_OPEN
    BUTTON      ID_FILE_SAVE
    SEPARATOR
    BUTTON      ID_EDIT_CUT
    BUTTON      ID_EDIT_COPY
    BUTTON      ID_EDIT_PASTE
    SEPARATOR
    BUTTON      ID_CAPTURE_START
    BUTTON      ID_CAPTURE_STOP
    BUTTON      ID_CAPTURE_SCROLL
    BUTTON      ID_CAPTURE_PARAMTERS
    SEPARATOR
    BUTTON      ID_CAMERA_LOADVIDEOPROCESSOR
    BUTTON      ID_CAMERA_PARAMETERS
    SEPARATOR
    BUTTON      ID_FILE_PRINT
    BUTTON      ID_APP_ABOUT
END


////////////////////////////////////////////////////////////////////////////
//
// Menu
//

IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "Ca&mera"
    BEGIN
        MENUITEM "&Load Video Processor...",    ID_CAMERA_LOADVIDEOPROCESSOR
        MENUITEM "Camera Parameters...",        ID_CAMERA_PARAMETERS
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About Larch...",             ID_APP_ABOUT
    END
END

IDR_LARCHTYPE MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
```

```
    BEGIN
        MENUITEM "&New\tCtrl+N",                ID_FILE_NEW
        MENUITEM "&Open...\tCtrl+O",            ID_FILE_OPEN
        MENUITEM "&Close",                      ID_FILE_CLOSE
        MENUITEM "&Save\tCtrl+S",               ID_FILE_SAVE
        MENUITEM "Save &As...",                 ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "Recent File",                 ID_FILE_MRU_FILE1, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       ID_APP_EXIT
    END
    POPUP "&Capture"
    BEGIN
        MENUITEM "&Start Capture",              ID_CAPTURE_START
        MENUITEM "&Stop Capture",               ID_CAPTURE_STOP
        MENUITEM "&Follow Capture",             ID_CAPTURE_SCROLL
        MENUITEM "Capture Parameters...",       ID_CAPTURE_PARAMTERS
    END
    POPUP "Ca&mera"
    BEGIN
        MENUITEM "&Load Video Processor...",    ID_CAMERA_LOADVIDEOPROCESSOR
        MENUITEM "Camera Parameters...",        ID_CAMERA_PARAMETERS
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               ID_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                ID_EDIT_CUT
        MENUITEM "&Copy\tCtrl+C",               ID_EDIT_COPY
        MENUITEM "&Paste\tCtrl+V",              ID_EDIT_PASTE
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Toolbar",                    ID_VIEW_TOOLBAR
        MENUITEM "&Status Bar",                 ID_VIEW_STATUS_BAR
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window",                 ID_WINDOW_NEW
        MENUITEM "&Cascade",                    ID_WINDOW_CASCADE
        MENUITEM "&Tile",                       ID_WINDOW_TILE_HORZ
        MENUITEM "&Arrange Icons",              ID_WINDOW_ARRANGE
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About Larch...",             ID_APP_ABOUT
    END
END


////////////////////////////////////////////////////////////////////////////
//
// Accelerator
//

IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE PURE
BEGIN
    "C",            ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    "N",            ID_FILE_NEW,            VIRTKEY, CONTROL, NOINVERT
```

```
    "O",            ID_FILE_OPEN,           VIRTKEY, CONTROL, NOINVERT
    "S",            ID_FILE_SAVE,           VIRTKEY, CONTROL, NOINVERT
    "V",            ID_EDIT_PASTE,          VIRTKEY, CONTROL, NOINVERT
    VK_BACK,        ID_EDIT_UNDO,           VIRTKEY, ALT, NOINVERT
    VK_DELETE,      ID_EDIT_CUT,            VIRTKEY, SHIFT, NOINVERT
    VK_DOWN,        ID_KEY_DOWN,            VIRTKEY, NOINVERT
    VK_END,         ID_KEY_END,             VIRTKEY, NOINVERT
    VK_F6,          ID_NEXT_PANE,           VIRTKEY, NOINVERT
    VK_F6,          ID_PREV_PANE,           VIRTKEY, SHIFT, NOINVERT
    VK_HOME,        ID_KEY_HOME,            VIRTKEY, NOINVERT
    VK_INSERT,      ID_EDIT_COPY,           VIRTKEY, CONTROL, NOINVERT
    VK_INSERT,      ID_EDIT_PASTE,          VIRTKEY, SHIFT, NOINVERT
    VK_LEFT,        ID_KEY_LEFT,            VIRTKEY, NOINVERT
    VK_NEXT,        ID_KEY_PAGEDOWN,        VIRTKEY, NOINVERT
    VK_PRIOR,       ID_KEY_PAGEUP,          VIRTKEY, NOINVERT
    VK_RIGHT,       ID_KEY_RIGHT,           VIRTKEY, NOINVERT
    VK_UP,          ID_KEY_UP,              VIRTKEY, NOINVERT
    "X",            ID_EDIT_CUT,            VIRTKEY, CONTROL, NOINVERT
    "Z",            ID_EDIT_UNDO,           VIRTKEY, CONTROL, NOINVERT
END


////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 217, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Larch"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "Larch Version 1.0",IDC_STATIC,40,10,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 1998",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "OK",IDOK,178,7,32,14,WS_GROUP
END

IDD_PARAMS DIALOGEX 0, 0, 202, 215
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Capture Parameters"
FONT 8, "MS Sans Serif"
BEGIN
    CTEXT           "Title",IDC_TITLE,5,6,190,8
    LTEXT           "Unused",IDC_TEXT1,15,24,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT1,149,21,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN1,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,21,11,14
    LTEXT           "Unused",IDC_TEXT2,15,45,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT2,149,42,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN2,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,42,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT3,15,66,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT3,149,63,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN3,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,63,11,14,WS_EX_STATICEDGE
```

```
    LTEXT           "Unused",IDC_TEXT4,15,87,124,9,WS_DISABLED
    EDITTEXT        IDC_EDIT4,149,84,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN4,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,84,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT5,15,108,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT5,149,105,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN5,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,105,11,14
    LTEXT           "Unused",IDC_TEXT6,15,129,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT6,149,126,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN6,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,126,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT7,15,150,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT7,149,146,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN7,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,146,11,14,WS_EX_STATICEDGE
    LTEXT           "Unused",IDC_TEXT8,15,170,124,8,WS_DISABLED
    EDITTEXT        IDC_EDIT8,149,168,36,14,WS_DISABLED
    CONTROL         "Spin1",IDC_SPIN8,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    WS_DISABLED,185,168,11,14,WS_EX_STATICEDGE
    DEFPUSHBUTTON   "OK",IDOK,15,192,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,135,192,50,14
END

IDD_CAMERA DIALOG DISCARDABLE  0, 0, 186, 114
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Camera Parameters"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT        IDC_EDITGAIN1,132,7,36,14,ES_AUTOHSCROLL
    CONTROL         "Spin1",IDC_SPINGAIN1,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    UDS_NOTHOUSANDS,168,7,11,14
    EDITTEXT        IDC_EDITGAIN2,132,28,36,14,ES_AUTOHSCROLL
    CONTROL         "Spin1",IDC_SPINGAIN2,"msctls_updown32",UDS_SETBUDDYINT |
                    UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS |
                    UDS_NOTHOUSANDS,168,28,11,14
    EDITTEXT        IDC_EDITOFFSET1,132,49,36,14,ES_AUTOHSCROLL
    CONTROL         "Spin1",IDC_SPINOFFSET1,"msctls_updown32",
                    UDS_SETBUDDYINT | UDS_ALIGNRIGHT | UDS_AUTOBUDDY |
                    UDS_ARROWKEYS | UDS_NOTHOUSANDS,168,49,11,14
    EDITTEXT        IDC_EDITOFFSET2,132,71,36,14,ES_AUTOHSCROLL
    CONTROL         "Spin1",IDC_SPINOFFSET2,"msctls_updown32",
                    UDS_SETBUDDYINT | UDS_ALIGNRIGHT | UDS_AUTOBUDDY |
                    UDS_ARROWKEYS | UDS_NOTHOUSANDS,168,71,11,14
    DEFPUSHBUTTON   "OK",IDOK,15,92,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,117,93,50,14
    LTEXT           "Gain Multiplier Channel 1 (x100)",IDC_STATICGAIN1,15,10,
                    114,8
    LTEXT           "Gain Multiplier Channel 2 (x100)",IDC_STATICGAIN2,15,31,
                    114,8
    LTEXT           "Offset Channel 1 (%)",IDC_STATICOFFSET1,15,52,114,8
    LTEXT           "Offset Channel 2 (%)",IDC_STATICOFFSET2,15,73,114,8
END
```

```
#ifndef _MAC
//////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904B0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "Larch MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "Larch\0"
            VALUE "LegalCopyright", "Copyright (C) 1998\0"
            VALUE "LegalTrademarks", "\0"
            VALUE "OriginalFilename", "Larch.EXE\0"
            VALUE "ProductName", "Larch Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

#endif    // !_MAC


//////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 210
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
```

```
    END

    IDD_PARAMS, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 196
        TOPMARGIN, 5
        BOTTOMMARGIN, 206
    END

    IDD_CAMERA, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 179
        TOPMARGIN, 7
        BOTTOMMARGIN, 107
    END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    IDR_MAINFRAME           "Larch"
    IDR_LARCHTYPE           "\nLarch\nLarch\nLarch Files
(*.cpt)\n.cpt\nLarch.Document\nLarch Document"
    ID_STATUS_CAPTURE       "CAPTURE"
    ID_STATUS_VALUE         "000"
END

STRINGTABLE PRELOAD DISCARDABLE
BEGIN
    AFX_IDS_APP_TITLE       "Larch"
    AFX_IDS_IDLEMESSAGE     "Ready"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_INDICATOR_EXT        "EXT"
    ID_INDICATOR_CAPS       "CAP"
    ID_INDICATOR_NUM        "NUM"
    ID_INDICATOR_SCRL       "SCRL"
    ID_INDICATOR_OVR        "OVR"
    ID_INDICATOR_REC        "REC"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW             "Create a new document\nNew"
    ID_FILE_OPEN            "Open an existing document\nOpen"
    ID_FILE_CLOSE           "Close the active document\nClose"
    ID_FILE_SAVE            "Save the active document\nSave"
    ID_FILE_SAVE_AS         "Save the active document with a new name\nSave As"
END
```

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT            "Display program information, version number and
copyright\nAbout"
    ID_APP_EXIT             "Quit the application; prompts to save
documents\nExit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1       "Open this document"
    ID_FILE_MRU_FILE2       "Open this document"
    ID_FILE_MRU_FILE3       "Open this document"
    ID_FILE_MRU_FILE4       "Open this document"
    ID_FILE_MRU_FILE5       "Open this document"
    ID_FILE_MRU_FILE6       "Open this document"
    ID_FILE_MRU_FILE7       "Open this document"
    ID_FILE_MRU_FILE8       "Open this document"
    ID_FILE_MRU_FILE9       "Open this document"
    ID_FILE_MRU_FILE10      "Open this document"
    ID_FILE_MRU_FILE11      "Open this document"
    ID_FILE_MRU_FILE12      "Open this document"
    ID_FILE_MRU_FILE13      "Open this document"
    ID_FILE_MRU_FILE14      "Open this document"
    ID_FILE_MRU_FILE15      "Open this document"
    ID_FILE_MRU_FILE16      "Open this document"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_NEXT_PANE            "Switch to the next window pane\nNext Pane"
    ID_PREV_PANE            "Switch back to the previous window pane\nPrevious
Pane"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_WINDOW_NEW           "Open another window for the active document\nNew
Window"
    ID_WINDOW_ARRANGE       "Arrange icons at the bottom of the window\nArrange
Icons"
    ID_WINDOW_CASCADE       "Arrange windows so they overlap\nCascade Windows"
    ID_WINDOW_TILE_HORZ     "Arrange windows as non-overlapping tiles\nTile
Windows"
    ID_WINDOW_TILE_VERT     "Arrange windows as non-overlapping tiles\nTile
Windows"
    ID_WINDOW_SPLIT         "Split the active window into panes\nSplit"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_EDIT_CLEAR           "Erase the selection\nErase"
    ID_EDIT_CLEAR_ALL       "Erase everything\nErase All"
    ID_EDIT_COPY            "Copy the selection and put it on the Clipboard\nCopy"
    ID_EDIT_CUT             "Cut the selection and put it on the Clipboard\nCut"
    ID_EDIT_FIND            "Find the specified text\nFind"
    ID_EDIT_PASTE           "Insert Clipboard contents\nPaste"
    ID_EDIT_REPEAT          "Repeat the last action\nRepeat"
```

```
    ID_EDIT_REPLACE         "Replace specific text with different text\nReplace"
    ID_EDIT_SELECT_ALL      "Select the entire document\nSelect All"
    ID_EDIT_UNDO            "Undo the last action\nUndo"
    ID_EDIT_REDO            "Redo the previously undone action\nRedo"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_VIEW_TOOLBAR         "Show or hide the toolbar\nToggle ToolBar"
    ID_VIEW_STATUS_BAR      "Show or hide the status bar\nToggle StatusBar"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCSIZE          "Change the window size"
    AFX_IDS_SCMOVE          "Change the window position"
    AFX_IDS_SCMINIMIZE      "Reduce the window to an icon"
    AFX_IDS_SCMAXIMIZE      "Enlarge the window to full size"
    AFX_IDS_SCNEXTWINDOW    "Switch to the next document window"
    AFX_IDS_SCPREVWINDOW    "Switch to the previous document window"
    AFX_IDS_SCCLOSE         "Close the active window and prompts to save the
documents"
END

STRINGTABLE DISCARDABLE
BEGIN
    AFX_IDS_SCRESTORE       "Restore the window to normal size"
    AFX_IDS_SCTASKLIST      "Activate Task List"
    AFX_IDS_MDICHILD        "Activate this window"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_CAPTURE_START        "Start capturing data\nStart Capture"
    ID_CAPTURE_STOP         "Stop capturing data\nStop Capture"
    ID_CAPTURE_SCROLL       "Scroll image to the most recent data\nFollow
Capture"
    ID_CAPTURE_PARAMTERS    "Sets the Capturing Paramters.\nCapture Parameters"
END

STRINGTABLE DISCARDABLE
BEGIN
    ID_CAMERA_LOADVIDEOPROCESSOR
                            "Loads a new Video Processor\nLoad Video Processor..."
    ID_CAMERA_PARAMETERS    "Set Camera Parameters...\nCamera Parameters"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
```

```
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif
#include "res\Larch.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#endif
/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

D.5.4 MFC Files

D.5.4.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__762CC6B4_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
#define AFX_STDAFX_H__762CC6B4_27B4_11D2_A11D_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN// Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxmt.h>         // MFC thread extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>// MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_STDAFX_H__762CC6B4_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
```

D.5.4.2 StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//  Larch.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

D.5.5 CLarchApp:CWinApp Class

D.5.5.1 Larch.h

```
// Larch.h : main header file for the LARCH application
//

#if !defined(AFX_LARCH_H__762CC6B2_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
#define AFX_LARCH_H__762CC6B2_27B4_11D2_A11D_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols
#include "CamInt.h"// Added by ClassView

/////////////////////////////////////////////////////////////////////////////
// CLarchApp:
// See Larch.cpp for the implementation of this class
//

class CLarchApp : public CWinApp
{
public:
    void UpdateAll();
    void StatusBarCapture(BOOL on);
    void StatusBarGrayValue(int value);
    void StatusBarText(LPSTR text);
public:
    void RemoveCaptureMessages();
    CCamInt *Camera;
    CLarchApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLarchApp)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    virtual int Run();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CLarchApp)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

extern CLarchApp theApp;
#define WM_1394RESET WM_USER
```

```
#define WM_CAMERASIG1 WM_USER+1

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif // !defined(AFX_LARCH_H__762CC6B2_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
```

D.5.5.2 Larch.cpp

```
// Larch.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Larch.h"

#include "Image.h"
#include "Imageline.h"
#include "MainFrm.h"
#include "ChildFrm.h"
#include "LarchCptDoc.h"
#include "LarchCptView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLarchApp

BEGIN_MESSAGE_MAP(CLarchApp, CWinApp)
    //{{AFX_MSG_MAP(CLarchApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLarchApp construction

CLarchApp::CLarchApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CLarchApp object

CLarchApp theApp;
```

```
/////////////////////////////////////////////////////////////////////////////
// CLarchApp initialization

BOOL CLarchApp::InitInstance()
{
    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();// Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();// Call this when linking to MFC statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("UofW-DALSA"));

    LoadStdProfileSettings();  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
        IDR_LARCHTYPE,
        RUNTIME_CLASS(CLarchCptDoc),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CLarchCptView));
    AddDocTemplate(pDocTemplate);

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    Camera = new CCamInt;

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
```

```
        return FALSE;

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CLarchApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
```

```
// CLarchApp commands

int CLarchApp::ExitInstance()
{
    // TODO: Add your specialized code here and/or call the base class

    delete Camera;

    return CWinApp::ExitInstance();
}

/* Update status bar message */
void CLarchApp::StatusBarText(LPSTR text)
{
    CMainFrame *wnd = (CMainFrame*)m_pMainWnd;

    wnd->StatusBarText(text);
    UpdateAll();
}

/* Update gray value indicator */
void CLarchApp::StatusBarGrayValue(int value)
{
    CMainFrame *wnd = (CMainFrame*)m_pMainWnd;

    wnd->StatusBarGrayValue(value);
}

/* Update CAPTURE indicator */
void CLarchApp::StatusBarCapture(BOOL on)
{
    CMainFrame *wnd = (CMainFrame*)m_pMainWnd;

    wnd->StatusBarCapture(on);
}

/* Update parent window */
void CLarchApp::UpdateAll()
{
    m_pMainWnd->UpdateWindow();
}

/* New Run member to stop capture if incoming data is too much */
#ifdef WIN32
int CLarchApp::Run()
{
    // TODO: Add your specialized code here and/or call the base class
    if (m_pMainWnd == NULL && AfxOleGetUserCtrl())
    {
        // Not launched /Embedding or /Automation, but has no main window!
        TRACE0("Warning: m_pMainWnd is NULL in CWinApp::Run - quitting
application.\n");
        AfxPostQuitMessage(0);
    }

    ASSERT_VALID(this);

    // for tracking the idle time state
    BOOL bIdle = TRUE;
```

```
    LONG lIdleCount = 0;
    LONG CaptureCount = 0;

    // acquire and dispatch messages until a WM_QUIT message is received.
    for (;;)
    {
        // phase1: check to see if we can do idle work
        while (bIdle &&
            !::PeekMessage(&m_msgCur, NULL, NULL, NULL, PM_NOREMOVE))
        {
            // call OnIdle while in bIdle state
            if (!OnIdle(lIdleCount++))
                bIdle = FALSE; // assume "no idle" state
        }

        CaptureCount = 0;
        // phase2: pump messages while available
        do
        {
            if
(::PeekMessage(&m_msgCur,NULL,WM_CAMERASIG1,WM_CAMERASIG1+1,PM_REMOVE)==TRUE)
            {
//              TRACE0("1");

                if (m_msgCur.message==WM_CAMERASIG1) Camera-
>Sig1((BYTE*)m_msgCur.wParam,(DWORD)m_msgCur.lParam);
                else
                {
                    // pump message, but quit on WM_QUIT
                    return ExitInstance();
                }
                CaptureCount++;
                /* If more than 20 messages came in without any other messages, */
                /*  stop the capture.  Chances are no other messages are being */
                /*  processed, including repaints and button processes. */
                if (CaptureCount > 20) theApp.Camera->StopCaptureNow();
            }
            else
            {
//              TRACE0("0");
                CaptureCount = 0;
                // pump message, but quit on WM_QUIT
                if (!PumpMessage())
                    return ExitInstance();

                // reset "no idle" state after pumping "normal" message
                if (IsIdleMessage(&m_msgCur))
                {
                    bIdle = TRUE;
                    lIdleCount = 0;
                }

            }

        } while (::PeekMessage(&m_msgCur, NULL, NULL, NULL, PM_NOREMOVE));
    }

    ASSERT(FALSE);  // not reachable
```

```
}
#endif

/* Remove messages from camera since the pointers can be invalid */
void CLarchApp::RemoveCaptureMessages()
{
    int count=0;

    while
(PeekMessage(&m_msgCur,NULL,WM_CAMERASIG1,WM_CAMERASIG1+1,PM_REMOVE)==TRUE)
    {
        count++;
    }

    TRACE1("Messages removed: %d\n",count);

    return;
}
```

D.5.6 CMainFrame Class

D.5.6.1 MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__762CC6B6_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
#define AFX_MAINFRM_H__762CC6B6_27B4_11D2_A11D_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    void StatusBarCapture(BOOL on);
    void StatusBarGrayValue(int value);
    void StatusBarText(LPSTR text);
    CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    LRESULT OnCameraSig1(WPARAM wParam, LPARAM lParam);
```

```
    LRESULT OnBusReset(WPARAM wParam, LPARAM lParam);
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnCameraLoadvideoprocessor();
    afx_msg void OnCameraParameters();
    afx_msg void OnUpdateCameraParameters(CCmdUI* pCmdUI);
    afx_msg void OnUpdateCameraLoadvideoprocessor(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_MAINFRM_H__762CC6B6_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
```

D.5.6.2 MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Larch.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(ID_CAMERA_LOADVIDEOPROCESSOR, OnCameraLoadvideoprocessor)
    ON_COMMAND(ID_CAMERA_PARAMETERS, OnCameraParameters)
    ON_UPDATE_COMMAND_UI(ID_CAMERA_PARAMETERS, OnUpdateCameraParameters)
```

```
    ON_UPDATE_COMMAND_UI(ID_CAMERA_LOADVIDEOPROCESSOR,
OnUpdateCameraLoadvideoprocessor)
    //}}AFX_MSG_MAP

/* New messages for application to process bus reset and incoming data from 1394
*/
    ON_MESSAGE(WM_1394RESET, OnBusReset)
    ON_MESSAGE(WM_CAMERASIG1, OnCameraSig1)

END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicators
    ID_STATUS_VALUE,
    ID_STATUS_CAPTURE
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    /* Create MDI parent window */
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    /* Create toolbar */
    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    /* Create Status bar */
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
```

```
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    int width;
    UINT dummy;

    m_wndStatusBar.GetPaneInfo( m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE),
dummy, dummy, width);
    m_wndStatusBar.SetPaneInfo( m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE),
ID_STATUS_VALUE, SBPS_NORMAL, width+2);
//  m_wndStatusBar.SetPaneText( m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE),
"0", TRUE);

    /* Turn off Capture indicator */
    StatusBarCapture(FALSE);
    /* Disable gray value output */
    StatusBarGrayValue(-1);

    return 0;
}

/* Add options to main window before MFC creation of it */
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    /* add horizontal and vertical scrolling bars to window */
    cs.style|=WS_HSCROLL|WS_VSCROLL;

    return CMDIFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

/* Set the text on the status bar */
void CMainFrame::StatusBarText(LPSTR text)
{
//  m_wndStatusBar.SetWindowText(text);
    m_wndStatusBar.SetPaneText(0,text,TRUE);
```

```
    m_wndStatusBar.UpdateWindow();
}

/* Update gray value display on status bar */
void CMainFrame::StatusBarGrayValue(int value)
{
    if (value<0 || value > 255)
    {
        /* if gray value is not valid, disable display on status bar */
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE), SBPS_DISABLED);
    }
    else
    {
        /* gray value is valid, enable display and show the value */
        char string[5];
        _itoa(value,string,10);
        m_wndStatusBar.SetPaneText(
m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE), string, TRUE);
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_VALUE), SBPS_NORMAL);
    }
}

/* Update the capture status on the status bar */
void CMainFrame::StatusBarCapture(BOOL on)
{
    if (on==TRUE)
    {
        /* If on enable the area */
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_CAPTURE), SBPS_NORMAL);
    }
    else
    {
        /* If off, disable it, gray it out */
        m_wndStatusBar.SetPaneStyle(
m_wndStatusBar.CommandToIndex(ID_STATUS_CAPTURE), SBPS_DISABLED);
    }
}

/* Handle 1394 BUS RESET message */
LRESULT CMainFrame::OnBusReset(WPARAM wParam, LPARAM lParam)
{
    TRACE0("BUS RESET!!!\n");
    /* Call camera class to handle bus reset */
    theApp.Camera->BusReset();

    return TRUE;
}

/* Handle incoming data message */
LRESULT CMainFrame::OnCameraSig1(WPARAM wParam, LPARAM lParam)
{
//  TRACE0("CAMERASIG1!!!\n");
    /* Call camera class to handle incoming data */
    theApp.Camera->Sig1((BYTE*)wParam,(DWORD)lParam);

    return TRUE;
```

```
}

/* Load a new video module (from menu) */
void CMainFrame::OnCameraLoadvideoprocessor()
{
    // TODO: Add your command handler code here
    theApp.Camera->LoadProcessor();
}

/* Set camera parameters (from menu) */
void CMainFrame::OnCameraParameters()
{
    // TODO: Add your command handler code here
    theApp.Camera->SetParams();
}

/* Always enable video module loader option */
void CMainFrame::OnUpdateCameraParameters(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(TRUE);
}

/* Always enable camera preferences option */
void CMainFrame::OnUpdateCameraLoadvideoprocessor(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(TRUE);
}
```

D.5.7 CChildFrame Class

D.5.7.1 ChildFrm.h

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_CHILDFRM_H__762CC6B8_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
#define AFX_CHILDFRM_H__762CC6B8_27B4_11D2_A11D_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

// Attributes
public:

// Operations
public:

// Overrides
```

```
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CChildFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CChildFrame)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_CHILDFRM_H__762CC6B8_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
```

D.5.7.2 ChildFrm.cpp

```
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "Larch.h"

#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
```

```
{
    // TODO: add member initialization code here

}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CMDIChildWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics

#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers
```

D.5.8 CLarchCptView Class

D.5.8.1 LarchCptView.h

```
// LarchCptView.h : interface of the CLarchCptView class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_LARCHCPTVIEW_H__762CC6BC_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
#define AFX_LARCHCPTVIEW_H__762CC6BC_27B4_11D2_A11D_0000C0059AB9__INCLUDED_

#include "LarchCptDoc.h"// Added by ClassView
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CLarchCptView : public CScrollView
{
protected: // create from serialization only
    CLarchCptView();
    DECLARE_DYNCREATE(CLarchCptView)
```

```
// Attributes
public:
    CLarchCptDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLarchCptView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
    protected:
    virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
    //}}AFX_VIRTUAL

// Implementation
public:
    void MoveToBottom(int start, int end);
    virtual ~CLarchCptView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CLarchCptView)
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnCaptureScroll();
    afx_msg void OnUpdateCaptureScroll(CCmdUI* pCmdUI);
    afx_msg void OnCaptureStart();
    afx_msg void OnUpdateCaptureStart(CCmdUI* pCmdUI);
    afx_msg void OnCaptureStop();
    afx_msg void OnUpdateCaptureStop(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFilePrint(CCmdUI* pCmdUI);
    afx_msg void OnKeyDown();
    afx_msg void OnKeyEnd();
    afx_msg void OnKeyHome();
    afx_msg void OnKeyLeft();
    afx_msg void OnKeyPagedown();
    afx_msg void OnKeyPageup();
    afx_msg void OnKeyRight();
    afx_msg void OnKeyUp();
    afx_msg void OnCaptureParamters();
    afx_msg void OnUpdateCaptureParamters(CCmdUI* pCmdUI);
    afx_msg void OnCameraLoadvideoprocessor();
    afx_msg void OnCameraParameters();
    afx_msg void OnUpdateCameraParameters(CCmdUI* pCmdUI);
    afx_msg void OnEditCopy();
    afx_msg void OnUpdateEditCopy(CCmdUI* pCmdUI);
    afx_msg void OnSize(UINT nType, int cx, int cy);
```

```
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL m_repaint;
    BOOL m_movetobottom;
};

#ifndef _DEBUG  // debug version in LarchCptView.cpp
inline CLarchCptDoc* CLarchCptView::GetDocument()
   { return (CLarchCptDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_LARCHCPTVIEW_H__762CC6BC_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
```

D.5.8.2 LarchCptView.cpp

```
// LarchCptView.cpp : implementation of the CLarchCptView class
//

#include "stdafx.h"
#include "Larch.h"
#include "Image.h"
#include "Imageline.h"
#include "LarchCptDoc.h"
#include "LarchCptView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView

IMPLEMENT_DYNCREATE(CLarchCptView, CScrollView)

BEGIN_MESSAGE_MAP(CLarchCptView, CScrollView)
    //{{AFX_MSG_MAP(CLarchCptView)
    ON_WM_ERASEBKGND()
    ON_WM_MOUSEMOVE()
    ON_WM_SETFOCUS()
    ON_COMMAND(ID_CAPTURE_SCROLL, OnCaptureScroll)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_SCROLL, OnUpdateCaptureScroll)
    ON_COMMAND(ID_CAPTURE_START, OnCaptureStart)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_START, OnUpdateCaptureStart)
    ON_COMMAND(ID_CAPTURE_STOP, OnCaptureStop)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_STOP, OnUpdateCaptureStop)
    ON_UPDATE_COMMAND_UI(ID_FILE_PRINT, OnUpdateFilePrint)
    ON_COMMAND(ID_KEY_DOWN, OnKeyDown)
    ON_COMMAND(ID_KEY_END, OnKeyEnd)
    ON_COMMAND(ID_KEY_HOME, OnKeyHome)
```

```
    ON_COMMAND(ID_KEY_LEFT, OnKeyLeft)
    ON_COMMAND(ID_KEY_PAGEDOWN, OnKeyPagedown)
    ON_COMMAND(ID_KEY_PAGEUP, OnKeyPageup)
    ON_COMMAND(ID_KEY_RIGHT, OnKeyRight)
    ON_COMMAND(ID_KEY_UP, OnKeyUp)
    ON_COMMAND(ID_CAPTURE_PARAMTERS, OnCaptureParamters)
    ON_UPDATE_COMMAND_UI(ID_CAPTURE_PARAMTERS, OnUpdateCaptureParamters)
    ON_COMMAND(ID_CAMERA_LOADVIDEOPROCESSOR, OnCameraLoadvideoprocessor)
    ON_COMMAND(ID_CAMERA_PARAMETERS, OnCameraParameters)
    ON_UPDATE_COMMAND_UI(ID_CAMERA_PARAMETERS, OnUpdateCameraParameters)
    ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateEditCopy)
    ON_WM_SIZE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView construction/destruction

CLarchCptView::CLarchCptView()
{
    // TODO: add construction code here
    m_repaint = FALSE;
}

CLarchCptView::~CLarchCptView()
{
}

BOOL CLarchCptView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CScrollView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView drawing

void CLarchCptView::OnDraw(CDC* pDC)
{
    CLarchCptDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    // TODO: add draw code for native data here
    pDoc->m_image->Paint(pDC);

    m_repaint = FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView diagnostics

#ifdef _DEBUG
void CLarchCptView::AssertValid() const
{
    CScrollView::AssertValid();
}
```

```
void CLarchCptView::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

CLarchCptDoc* CLarchCptView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CLarchCptDoc)));
    return (CLarchCptDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CLarchCptView message handlers

/* Erase background where there is no image */
BOOL CLarchCptView::OnEraseBkgnd(CDC* pDC)
{
    // TODO: Add your message handler code here and/or call default

    CBrush br( GetSysColor( COLOR_WINDOW ) );
    FillOutsideRect( pDC, &br );

    return TRUE;                    // Erased
}

/* Update gray value of where the mouse pointer is on the image */
void CLarchCptView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CPoint scroll =GetScrollPosition();

    scroll = scroll + point;

//  TRACE2("Point: %d, %d\n",scroll.x,scroll.y);

    CSize size=GetTotalSize();

    if (scroll.x <= size.cx && scroll.y <= size.cy)
    {
        CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();
        int value = pDoc->m_image->GetValue(scroll.x,scroll.y);
        theApp.StatusBarGrayValue(value);
    }
    else
    {
        theApp.StatusBarGrayValue(-1);
    }

    CScrollView::OnMouseMove(nFlags, point);
}

/* When focus changes, update capture status */
void CLarchCptView::OnSetFocus(CWnd* pOldWnd)
{
    CScrollView::OnSetFocus(pOldWnd);

    // TODO: Add your message handler code here
```

```
    if (theApp.Camera)
    {
        theApp.StatusBarCapture(theApp.Camera->CanStopCapture(GetDocument()));
    }
    else
    {
        theApp.StatusBarCapture(FALSE);
    }
}

/* Set scroll bar sizes for initial update */
void CLarchCptView::OnInitialUpdate()
{
    CScrollView::OnInitialUpdate();

    // TODO: Add your specialized code here and/or call the base class

    m_movetobottom=FALSE;

    CLarchCptDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    /* Update scroll bars */
    SetScrollSizes(MM_TEXT, pDoc->m_image->GetScrollSize());
}

/* Update follow capture flag */
void CLarchCptView::OnCaptureScroll()
{
    // TODO: Add your command handler code here
    m_movetobottom = !m_movetobottom;
}

/* Update capture flag button */
void CLarchCptView::OnUpdateCaptureScroll(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(TRUE);
    pCmdUI->SetCheck(m_movetobottom);
}

/* Process capture start command */
void CLarchCptView::OnCaptureStart()
{
    // TODO: Add your command handler code here
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    pDoc->StartCapture();
    theApp.StatusBarCapture(theApp.Camera->CanStopCapture(pDoc));
}

/* Update capture start button */
void CLarchCptView::OnUpdateCaptureStart(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    pCmdUI->Enable(theApp.Camera->CanStartCapture(pDoc,0 /*pdoc something*/));
}
```

```
/* Process capture stop command */
void CLarchCptView::OnCaptureStop()
{
    // TODO: Add your command handler code here
    theApp.Camera->StopCapture(GetDocument());
    theApp.StatusBarCapture(theApp.Camera->CanStopCapture(GetDocument()));
}

/* Update capture stop button */
void CLarchCptView::OnUpdateCaptureStop(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(theApp.Camera->CanStopCapture(GetDocument()));
}

/* Disable printing */
void CLarchCptView::OnUpdateFilePrint(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(FALSE);
}

/* Move display pointer down a 10th of the client height for down key */
void CLarchCptView::OnKeyDown()
{
    // TODO: Add your command handler code here
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y+=rect.Height()/10;
    ScrollToPosition(point);
}

/* Move to bottom for end key */
void CLarchCptView::OnKeyEnd()
{
    // TODO: Add your command handler code here
    CSize size=GetTotalSize();
    CPoint point(0,size.cy);
    ScrollToPosition(point);
}

/* Move to top for home key */
void CLarchCptView::OnKeyHome()
{
    // TODO: Add your command handler code here
    CPoint point(0,0);
    ScrollToPosition(point);
}

/* Move display pointer left a 10th of the client width for left key */
void CLarchCptView::OnKeyLeft()
{
    // TODO: Add your command handler code here
    CPoint point = GetScrollPosition();
    CRect rect;
```

```
    GetClientRect(&rect);

    point.x-=rect.Width()/10;
    ScrollToPosition(point);
}

/* Move display pointer up client height for page up key */
void CLarchCptView::OnKeyPagedown()
{
    // TODO: Add your command handler code here
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y+=rect.Height();
    ScrollToPosition(point);
}

/* Move display pointer down client height for page down key */
void CLarchCptView::OnKeyPageup()
{
    // TODO: Add your command handler code here
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y-=rect.Height();
    ScrollToPosition(point);
}

/* Move display pointer right a 10th of the client width for righ */
void CLarchCptView::OnKeyRight()
{
    // TODO: Add your command handler code here
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.x+=rect.Width()/10;
    ScrollToPosition(point);
}

/* Move display pointer up a 10th of the client height for up key */
void CLarchCptView::OnKeyUp()
{
    // TODO: Add your command handler code here
    CPoint point = GetScrollPosition();
    CRect rect;

    GetClientRect(&rect);

    point.y-=rect.Height()/10;
    ScrollToPosition(point);
}
```

```
/* Update document view and set scroll bar sizes */
void CLarchCptView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
{
    // TODO: Add your specialized code here and/or call the base class
    CLarchCptDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    SetScrollSizes(MM_TEXT, pDoc->m_image->GetScrollSize());
}

/* Update display and move to bottom if necessary */
void CLarchCptView::MoveToBottom(int start, int end)
{
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    SetScrollSizes(MM_TEXT, pDoc->m_image->GetScrollSize());

    /* Get image size */
    CSize size=GetTotalSize();
    CRect rect,work;
    GetClientRect(&rect);

    /* Figure out what portions of the image must be updated */
    if (start<end)
    {
        /* Update a middle portion of the image */
        work.right=rect.right;
        work.left=rect.left;
        work.top=start;
        work.bottom=end;
        InvalidateRect(work,FALSE);
    }
    else
    {
        /* Update the top and bottom segments of the image */
        work.right=rect.right;
        work.left=rect.left;
        work.top=start;
        work.bottom=size.cy;
        InvalidateRect(work,FALSE);
        work.right=rect.right;
        work.left=rect.left;
        work.top=0;
        work.bottom=end;
        InvalidateRect(work,FALSE);
    }

    /* If set, move the display to the bottom to the current capture line */
    if (m_movetobottom==TRUE)
    {
        CPoint point = GetScrollPosition();

        point.y = max(0,end-rect.Height());

        ScrollToPosition(point);
    }

}
```

```
/* Open dialog for modifying video processor parameters */
void CLarchCptView::OnCaptureParamters()
{
    // TODO: Add your command handler code here
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    pDoc->CaptureParameters();
}

/* Update UI settings for button */
void CLarchCptView::OnUpdateCaptureParamters(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

    pCmdUI->Enable(theApp.Camera->CanStartCapture(pDoc,0));
}

/* Open dialog for loading video processor module */
void CLarchCptView::OnCameraLoadvideoprocessor()
{
    // TODO: Add your command handler code here
    theApp.Camera->LoadProcessor();
}

/* Open camera settings dialog box */
void CLarchCptView::OnCameraParameters()
{
    // TODO: Add your command handler code here
    theApp.Camera->SetParams();
}

/* Update UI settings for button */
void CLarchCptView::OnUpdateCameraParameters(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(theApp.Camera->CanStartCapture(GetDocument(),0));
}

/* Copy capture data to windows clipboard */
void CLarchCptView::OnEditCopy()
{
    // TODO: Add your command handler code here
    CDC *pDC;

    /* Try to open clipboard */
    if (OpenClipboard())
    {
        /* Clean it first */
        EmptyClipboard();

        /* get a draw context */
        pDC = GetDC();

        /* locate our parent document */
        CLarchCptDoc* pDoc = (CLarchCptDoc*)GetDocument();

        /* Call sub class */
        pDoc->m_image->CopyAll(pDC);
```

```
        /* Release draw context */
        ReleaseDC(pDC);

        /* Close clipboard */
        CloseClipboard();
    }
    else
    {
        /* Beep if we can't open the clipboard */
        MessageBeep(0);
    }

}

/* Update copy all button if sub class says we can */
void CLarchCptView::OnUpdateEditCopy(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
    pCmdUI->Enable(TRUE);
}

/* When window is resized, set a flag to indicate a repaint */
void CLarchCptView::OnSize(UINT nType, int cx, int cy)
{
    m_repaint = TRUE;
    CScrollView::OnSize(nType, cx, cy);

}
```

D.5.9 CLarchCptDoc Class

D.5.9.1 LarchCptDoc.h

```
// LarchCptDoc.h : interface of the CLarchCptDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_LARCHCPTDOC_H__762CC6BA_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
#define AFX_LARCHCPTDOC_H__762CC6BA_27B4_11D2_A11D_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CLarchCptDoc : public CDocument
{
protected: // create from serialization only
    CLarchCptDoc();
    DECLARE_DYNCREATE(CLarchCptDoc)

// Attributes
public:

// Operations
public:
```

```
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CLarchCptDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    virtual void OnCloseDocument();
    virtual void SetTitle(LPCTSTR lpszTitle);
    //}}AFX_VIRTUAL

// Implementation
public:
    void UpdateForCapture(int start, int end);
    void CaptureParameters();
    void StartCapture();
    void TAddData(BYTE *data, int total);
    BOOL InitDocument();
    int m_linelength;
    CImage *m_image;
    virtual ~CLarchCptDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CLarchCptDoc)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    BOOL SetProcessorParams();
    BOOL GetProcessorParams();
    void (*m_processorsd)(void *);
    int m_params[8];
    BYTE m_coded[8];
    BYTE *m_processorpb;
    void (*m_processorgpb)(void *,BYTE *);
    DWORD (*m_processorg)(void *,int *, BYTE *);
    void (*m_processorgdp)(void *, int *);
    CString m_oldtitle;
    CString m_newtitle;
    char m_processorname[50];
//  BOOL m_processorad;
    BYTE *(*m_processorgfb)(int *);
    BOOL (*m_processorad)(void *, BYTE **, int *, void *);
    BYTE *m_processorpd;
    HINSTANCE m_processordll;
    int m_tsynced;
    int m_tcurrentoffset;
    int m_tcurrentline;
};

/////////////////////////////////////////////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_LARCHCPTDOC_H__762CC6BA_27B4_11D2_A11D_0000C0059AB9__INCLUDED_)
```

D.5.9.2 LarchCptDoc.cpp

```
// LarchCptDoc.cpp : implementation of the CLarchCptDoc class
//

#include "stdafx.h"
#include "Larch.h"
#include "Image.h"
#include "Imageline.h"
#include "LarchCptDoc.h"
#include "LarchCptView.h"
#include "Params.h"

#include "Processor.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc

IMPLEMENT_DYNCREATE(CLarchCptDoc, CDocument)

BEGIN_MESSAGE_MAP(CLarchCptDoc, CDocument)
    //{{AFX_MSG_MAP(CLarchCptDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc construction/destruction

CLarchCptDoc::CLarchCptDoc()
{
    // TODO: add one-time construction code here
    m_image = NULL;
    m_linelength = 0;
    m_processorpd = NULL;
    m_processorpb = NULL;
}

CLarchCptDoc::~CLarchCptDoc()
{
}

BOOL CLarchCptDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
```

```
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    /* Set up an empty document */
    m_image = NULL;
    m_linelength = 508; /* any size really */

    int check=0;
//  m_processordll = AfxLoadLibrary("..\\Lineup\\Debug\\Lineup.vpr");
//  m_processordll = AfxLoadLibrary("..\\Focus\\Debug\\Focus.vpr");
//  m_processordll = AfxLoadLibrary("..\\Minmax\\Debug\\Minmax.vpr");

    /* get DLL entry point */
    m_processordll = AfxLoadLibrary(theApp.Camera->GetDefaultProcessor());

    /* Make sure it exists */
    if (!m_processordll) return FALSE;
    int (*pds)(void);
    int (*id)(BYTE *);
    int (*gt)(char *,int);
    /* Set local pointer to setting up Private Data Size function */
    pds = (int (*)(void))GetProcAddress(m_processordll,"PrivateDataSize");
    /* Set local pointer to Initialize Data function */
    id = (int (*)(BYTE*))GetProcAddress(m_processordll,"InitData");
    /* Set pointer to Start Data function */
    m_processorsd = (void (*)(void *))GetProcAddress(m_processordll,"StartData");
    /* Set pointer to Add Data function */
    m_processorad = (BOOL (*)(void *, BYTE **, int *, void
*))GetProcAddress(m_processordll,"AddData");
    /* Set pointer to Get FPGA Bitmap function */
    m_processorgfb = (BYTE* (*)(int
*))GetProcAddress(m_processordll,"GetFPGABitmap");
    /* Set local pointer to Get Title function */
    gt = (int (*)(char *,int))GetProcAddress(m_processordll,"GetTitle");
    /* Set pointer to Get Default Parameters function */
    m_processorgdp = (void (*)(void *,int
*))GetProcAddress(m_processordll,"GetDefaultParams");
    /* Set pointer to Generate function */
    m_processorg = (DWORD (*)(void *,int *, BYTE
*))GetProcAddress(m_processordll,"Generate");
    /* Set pointer to Get Parameter function */
    m_processorgpb = (void (*)(void *,BYTE
*))GetProcAddress(m_processordll,"GetParamBlock");

    /* Make sure all pointers are valid before continuing */
    if (pds && id && m_processorad && m_processorgfb && gt && m_processorgdp &&
m_processorg && m_processorgpb)
    {
        /* Allocate private data size */
        /*  DLL can't have it's one memory management, we must provide it */
        int size = pds();
        if (size)
        {
            m_processorpd = new BYTE[size];
            ASSERT(m_processorpd);
        }
        /* Call initialize data function with more memory */
```

```
        size = id(m_processorpd);
        if (size)
        {
            m_processorpb = new BYTE[size];
            ASSERT(m_processorpb);
        }
        /* Set processor name */
        gt(m_processorname,sizeof(m_processorname)-1);
        GetProcessorParams();
        SetProcessorParams();
        /* Set new line length based on the the paramters */
        m_linelength = m_processorg(m_processorpd,m_params,m_coded);
        m_processorgpb(m_processorpd,m_processorpb);

        check++;
    }

    /* If anything fales, release memory */
    if (!check)
    {
        if (m_processorpd) delete m_processorpd;
        if (m_processorpb) delete m_processorpb;
        AfxFreeLibrary(m_processordll);

        return FALSE;
    }

    /* Initialize the document */
    if (InitDocument()==FALSE)
    {
        return FALSE;
    }

    return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc serialization

/* Don't support saving and loading yet */
void CLarchCptDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc diagnostics

#ifdef _DEBUG
void CLarchCptDoc::AssertValid() const
```

```
{
    CDocument::AssertValid();
}

void CLarchCptDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CLarchCptDoc commands

/* Initialize document and image data */
BOOL CLarchCptDoc::InitDocument()
{
    TRACE0("CLarchCptDoc::InitDocument()\n");

    m_tcurrentline = 0;
    m_tcurrentoffset = 0;
    m_tsynced = 0;

    /* Create image if line length is not zero */
    if (m_linelength >= 0 )
    {
        m_image = new CImage(m_linelength);
    }
    else
    {
        m_image = NULL;

        return FALSE;
    }

    /* Set window name */
    m_newtitle = m_oldtitle + " (" + m_processorname + ")";
    CDocument::SetTitle(m_newtitle);

    return TRUE;
}

/* Close document */
void CLarchCptDoc::OnCloseDocument()
{
    // TODO: Add your specialized code here and/or call the base class
    /* Destory image */
    if (m_image != NULL)
    {
        delete m_image;
    }

    /* Delete processor data */
    m_image = NULL;

    if (m_processorpd) delete m_processorpd;
    if (m_processorpb) delete m_processorpb;

    /* Unload DLL */
    AfxFreeLibrary(m_processordll);
```

```
    /* Go on with MFC function */
    CDocument::OnCloseDocument();
}

/* Temporary function, but it works now */
/* Process incoming data from camera to video processor */
void CLarchCptDoc::TAddData(BYTE *data, int total)
{
    BOOL go=TRUE;
    int left = total;
    BYTE *point = data;
    lineinfo li;
    CImageline *line=NULL;
    int start = m_tcurrentline;
    int end;

    /* Keep going until all data has been processed */
    while (go == TRUE)
    {
        /* Get pointer to new line and expand if necessary */
        line = new CImageline(m_image, m_tcurrentline, TRUE);
        /* Create structure to be passed to decompressor */
        li.data = line->m_pointer;
        li.line = m_tcurrentline;
        /* Decompress a line */
        go = m_processorad(m_processorpd,&point,&left,&li);
        delete line;

        /* 4096 lines on the drum...  If we don't have this, we will run out */
        /*  of memory REALLY fast! */
        m_tcurrentline = (li.line) % 4096;
    }

    end = m_tcurrentline;

    /* Update display of what was changed/added */
    UpdateForCapture(start,end);
}

/* Update all views of the document */
void CLarchCptDoc::UpdateForCapture(int start, int end)
{
    POSITION pos = GetFirstViewPosition();

    while (pos != NULL)
    {
        CLarchCptView* pView = (CLarchCptView*) GetNextView(pos);
        pView->MoveToBottom(start, end);
    }
}

/* Set the document based on the title of the video processor */
void CLarchCptDoc::SetTitle(LPCTSTR lpszTitle)
{
    // TODO: Add your specialized code here and/or call the base class

    CString oldtitle(lpszTitle);
```

```
    if (m_oldtitle != oldtitle)
    {
        m_oldtitle = oldtitle;
        m_newtitle = m_oldtitle + " (" + m_processorname + ")";
    }

    CDocument::SetTitle(m_newtitle);
}

/* Start capture if it can */
void CLarchCptDoc::StartCapture()
{
    BYTE *data;
    int size;

    /* Get size and pointer of FPGA bitstream */
    data = m_processorgfb(&size);

    /* make sure we have a valid FPGA bitstream */
    if (data)
    {
        /* Program the FPGA */
        if (theApp.Camera->LoadFPGA(data,size,m_processorname)==TRUE)
        {
            /* Processor start incoming data */
            m_processorsd(m_processorpd);
//          TRACE0("Sending Coded\n");
            /* Set coded data */
            theApp.Camera->SendCoded(m_coded);
//          TRACE0("Starting Capture\n");
            /* Start capture at camera layer */
            theApp.Camera->StartCapture(this);
        }
    }

    return;
}

/* Get new paramters from user, called by UI */
void CLarchCptDoc::CaptureParameters()
{
    /* Call dialog function */
    CParams *proc = new CParams(NULL,m_processorpb);

    /* Pass parameters to the dialog */
    proc->SetParams(m_params);
    /* Proceed through dialog to process user functions */
    proc->DoModal();
    /* Get parameters from the dialog */
    proc->GetParams(m_params);

    delete proc;

    /* Save processor parameters in registry immediately */
    SetProcessorParams();

    /* Check new line length, set FPGA encoded bytes */
    int a = m_processorg(m_processorpd,m_params,m_coded);
    TRACE1("New linelength=%d\n",a);
```

```
    /* If different from before, generate a new image */
    if (a != m_linelength)
    {
        m_linelength = a;
        /* Delete old image */
        if (m_image) delete m_image;

        /* Create new image */
        m_image = new CImage(m_linelength);

        /* Update all image displays */
        UpdateForCapture(0,0xffffffff);
    }
}


/* Get processor defaults from registry */
BOOL CLarchCptDoc::GetProcessorParams()
{
    char temp[20];
    int i,j;

    for (i=0;i<GETPARAMS_MAX;i++)
    {
        /* Generate KEY name */
        sprintf(temp,"P%d",i);
        j=theApp.GetProfileInt(m_processorname,temp,-32768);
        if (j==-32768)
        {
            TRACE1("No parameters stored for %s, using
defaults.\n",m_processorname);
            /* Use defaults if nothing found in registry */
            m_processorgdp(m_processorpd,m_params);

            return TRUE;
        }
        m_params[i]=j;

    }

    return TRUE;
}

/* Set processor defaults into the registry */
BOOL CLarchCptDoc::SetProcessorParams()
{
    char temp[20];
    int i;

    for (i=0;i<GETPARAMS_MAX;i++)
    {
        sprintf(temp,"P%d",i);
        if (theApp.WriteProfileInt(m_processorname,temp,m_params[i])==FALSE)
return FALSE;
    }

    return TRUE;
}
```

D.5.10 CamInt Class

D.5.10.1 CamInt.h

```
// CamInt.h: interface for the CCamInt class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CAMINT_H__0B508AD1_296A_11D2_A11E_0000C0059AB9__INCLUDED_)
#define AFX_CAMINT_H__0B508AD1_296A_11D2_A11E_0000C0059AB9__INCLUDED_

#include "Physical.h"// Added by ClassView
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CCamInt
{
public:
    void SetParams();
    CString GetDefaultProcessor();
    BOOL LoadProcessor();
    void StopCaptureNow();
    BOOL SendCoded(BYTE *coded);
    BOOL LoadFPGA(BYTE *bitmap, int size, char *name);
    void Sigl(BYTE *indata, DWORD bytesreceived);
    void BusReset();
    BOOL CanStartCapture(CDocument *doc, int width);
    BOOL CanStopCapture(CDocument *doc);
    BOOL StopCapture(CDocument *doc);
    BOOL StartCapture(CDocument *doc);
    CCamInt();
    virtual ~CCamInt();

private:
    DWORD m_bytesin;
    time_t m_lasttime;
    int m_params[4];
    void SetDefaults();
    void GetDefaults();
    CString m_defaultprocessorpath;
    CString m_defaultprocessorname;
    char m_currentprocessor[50];
    DWORD m_eventcount;
    /* Create physical interface object */
    CPhysical m_physical;
    CDocument *m_capdoc;
    int m_units;
};

#endif //
!defined(AFX_CAMINT_H__0B508AD1_296A_11D2_A11E_0000C0059AB9__INCLUDED_)
```

D.5.10.2 CamInt.cpp

```
// CamInt.cpp: implementation of the CCamInt class.
```

```
//
//////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Larch.h"
#include "CamInt.h"

#include "Image.h"
#include "Imageline.h"
#include "MainFrm.h"
#include "ChildFrm.h"
#include "LarchCptDoc.h"
#include "LarchCptView.h"

#include "CamParams.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////////

CCamInt::CCamInt()
{
    /* Setup some variable */
    m_capdoc = NULL;
    /* Physical object created already, should have the number of cameras */
    m_units = m_physical.NumberOfCameras();
    m_eventcount = 0;

    /* Get the defaults of the Video processor */
    GetDefaults();
}

CCamInt::~CCamInt()
{
}

BOOL CCamInt::StartCapture(CDocument * doc)
{
    /* Start capture if there is a document available */
    if (m_capdoc != NULL)
    {
        return FALSE;
    }

    CWaitCursor wait;

    m_capdoc = doc;
    m_lasttime = 0;
    /* Call the physical layer */
    m_physical.StartCapture();
    m_eventcount = 0;

    return TRUE;
```

```
}

BOOL CCamInt::StopCapture(CDocument * doc)
{
    /* Stop capture if there is a document available */
    if (m_capdoc != doc)
    {
        return FALSE;
    }

    CWaitCursor wait;

    m_physical.StopCapture();
    /* Must remove messages, or else we can crash with NULL pointers in the kernel
*/
    theApp.RemoveCaptureMessages();
    m_capdoc = NULL;

    return TRUE;
}

void CCamInt::StopCaptureNow()
{
    /* Stop capture bypassing document check */
    StopCapture(m_capdoc);
}

/* Update UI Capture stop button */
BOOL CCamInt::CanStopCapture(CDocument * doc)
{
    if (m_capdoc != doc) return FALSE;

    return TRUE;
}

/* Update UI Capture start button */
BOOL CCamInt::CanStartCapture(CDocument * doc, int width)
{
    if (m_capdoc != NULL) return FALSE;

    if (m_units < 1) return FALSE;

    return TRUE;
}

/* Called by main message system to reset bus */
void CCamInt::BusReset()
{
    CWaitCursor wait;

    /* Remove all unprocessed messages */
    theApp.RemoveCaptureMessages();
    /* Wait a sec */
    Sleep(1000);
    /* Get the number of cameras */
    m_units = m_physical.NumberOfCameras();

    /* Set the current processor off, so the FPGA algorithm will be reloaded */
    m_currentprocessor[0]=0;
```

```
    /* More messages may have appeared, clear them */
    theApp.RemoveCaptureMessages();
}

/* Process message signal generated by other thread */
void CCamInt::Sig1(BYTE *indata, DWORD bytesreceived)
{
    time_t ltime;

    /* Find out what the average throughput */
    time( &ltime );

    m_eventcount++;
    if ((m_eventcount & 15)==0) TRACE2("CCamInt::Sig1(%d) Got %d
bytes.\n",m_eventcount,bytesreceived);

    if (m_lasttime!=0)
    {
        if ((m_eventcount & 63)==0)
        {
            TRACE1("Average throughput: %lfkb/s\n",(double)m_bytesin/(ltime-
m_lasttime)/1024.0);
            m_bytesin = 0;
            m_lasttime = ltime;
        }

    }
    else
    {
        m_lasttime = ltime;
        m_bytesin = 0;
    }
    m_bytesin += bytesreceived;

    /* Record incoming data for debugging */
    /*
    FILE *out;
    out = fopen("e:\\trace.dat","wb");
    fwrite(indata,1,bytesreceived,out);
    fclose(out);
    */

    if (m_capdoc)
    {
        /* Get document pointer */
        CLarchCptDoc *doc = (CLarchCptDoc*) m_capdoc;

        /* Stop capture immediately for debugging */
//      StopCaptureNow();

        /* Call the routine to process the data */
        doc->TAddData(indata,bytesreceived);
    }

    /* Output some of the data for debugging */
/*
    int i;
    for (i=0;i<16;i++)
    {
```

```
        TRACE3("%d,%2x%2x\n",indata[i*4+2],indata[i*4],indata[i*4+1]);
    }
*/

}

/* Load the FPGA algorithm to the camera */
BOOL CCamInt::LoadFPGA(BYTE *bitmap, int size, char *name)
{
    /* If the algorithm is already loaded, just send the parameters */
    if (strcmp(name,m_currentprocessor)==0)
    {
        m_physical.SendParams(m_params);
        return TRUE;
    }
    /* Use physical layer to load the bitmap */
    if (m_physical.LoadFPGA(bitmap,size)==FALSE)
    {
        return FALSE;
    }
    /* Remember the name of the processor */
    strcpy(m_currentprocessor,name);
    /* Send the parameters */
    m_physical.SendParams(m_params);

    return TRUE;
}


BOOL CCamInt::SendCoded(BYTE *coded)
{
//  BYTE test[32] = "This is a test of the fpga RAM!";

    // RAM data for Fuzzy algorithm
    BYTE test[32] = {
          0xf8, 0xfc, 0xf8, 0xfc, 0xf8, 0xfc, 0xf8, 0xfc,
          0xf8, 0xfc, 0xf8, 0xfc, 0xf8, 0xfc, 0xf8, 0xfc,
          0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfe, 0xfc, 0xfe,
          0xfc, 0xfe, 0xfc, 0xfe, 0xfc, 0xff, 0xfc, 0xff
      };

/*
    // All zeros
    BYTE test[32] = {
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
      };
*/
/*
    // All ones
    BYTE test[32] = {
          0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
          0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
          0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
          0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
      };
*/
```

```
/*
    // Something else
    int i;
    for (i=0;i<32;i++)
        test[i]=0x80;
*/

    /* Send LUT to FPGA */
    m_physical.SendLUT(test);

    /* Send parameters about algorithm */
    return m_physical.SendCoded(coded);
}

/* Let the user select a processor from the disk */
BOOL CCamInt::LoadProcessor()
{
    /* Create a file dialog box to look for vpr files */
    CFileDialog dialog( TRUE, "vpr", NULL,
        OFN_HIDEREADONLY | OFN_EXPLORER | OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST,
        "Video Processor Files (*.vpr)|*.vpr||", NULL );

    /* Set the title */
    CString title("Load Video Processor");
    dialog.m_ofn.lpstrTitle = title;

    /* Call the dialog box function */
    if (dialog.DoModal()==IDOK)
    {
//      theApp.UpdateAll();
        int check=0;
        CString file;
        char pn[50];
//      m_defaultprocessorpath=dialog.GetPathName();
        file=dialog.GetPathName();
        /* Try to open the DLL */
        HINSTANCE dll;
        dll = AfxLoadLibrary(file);
        if (dll)
        {
            /* Get DLL name and path and save it as default for the application */
            int (*gt)(char *,int);
            gt = (int (*)(char *,int))GetProcAddress(dll,"GetTitle");
            if (gt)
            {
                gt(pn,sizeof(pn)-1);
                m_defaultprocessorpath=file;
                m_defaultprocessorname=CString(pn);

                SetDefaults();
            }
            else check++;

            AfxFreeLibrary(dll);
        }
        else
        {
            check++;
```

```
        }

        /* If it didn't work, tell the use and do nothing */
        if (check)
        {
            AfxMessageBox((CString)"Could not load '"+file+"'.",MB_ICONSTOP);

            return FALSE;
        }
    }

    return TRUE;
}

/* Return the deafult processor path */
CString CCamInt::GetDefaultProcessor()
{
    return m_defaultprocessorpath;
}

/* Get the camera defaults from the registry */
void CCamInt::GetDefaults()
{
    CString s1,s2;

    /* Default processor name and path */
    s1 = theApp.GetProfileString("Defaults","ProcessorName","");
    s2 = theApp.GetProfileString("Defaults","ProcessorPath","");
    /* Gain and offset of the two channels */
    m_params[0] = theApp.GetProfileInt("Defaults","Gain1",350);
    m_params[1] = theApp.GetProfileInt("Defaults","Gain2",133);
    m_params[2] = theApp.GetProfileInt("Defaults","Offset1",100);
    m_params[3] = theApp.GetProfileInt("Defaults","Offset2",32);

    /* If there is no default processor do nothing, wait for user to pick one */
    if (s1=="" || s2=="")
    {
//      LoadProcessor();
    }
    else
    {
        m_defaultprocessorname = s1;
        m_defaultprocessorpath = s2;
    }
}

/* Set the camera defaults to the registry */
void CCamInt::SetDefaults()
{

theApp.WriteProfileString("Defaults","ProcessorName",m_defaultprocessorname);

theApp.WriteProfileString("Defaults","ProcessorPath",m_defaultprocessorpath);
    theApp.WriteProfileInt("Defaults","Gain1",m_params[0]);
    theApp.WriteProfileInt("Defaults","Gain2",m_params[1]);
    theApp.WriteProfileInt("Defaults","Offset1",m_params[2]);
    theApp.WriteProfileInt("Defaults","Offset2",m_params[3]);
}
```

```
/* Dialog handle for camera parameters */
void CCamInt::SetParams()
{
    /* Create dialog */
    CCamParams *proc = new CCamParams(NULL);

    /* Set parameters */
    proc->SetParams(m_params);
    /* Do dialog */
    proc->DoModal();
    /* Get those parameters */
    proc->GetParams(m_params);

    /* Send the to the camera */
    m_physical.SendParams(m_params);

    delete proc;
}
```

D.5.11 Physical Class

D.5.11.1 Physical.h

```
// Physical.h: interface for the CPhysical class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_PHYSICAL_H__0B508AD2_296A_11D2_A11E_0000C0059AB9__INCLUDED_)
#define AFX_PHYSICAL_H__0B508AD2_296A_11D2_A11E_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include <papi.h>
#include <Exports.h>

struct microraw
{
    WORD length;
    WORD data[29];
};

struct microform
{
    WORD length;
    BYTE to;
    BYTE from;
    BYTE type;
    BYTE data[58-3];
};

union micromess
{
    struct microraw raw;
    struct microform form;
};
```

```
class CPhysical
{
public:
    BOOL SendLUT(BYTE *data);
    BOOL SendParams(int *params);
    BOOL SendCoded(BYTE *coded);
    BOOL LoadFPGA(BYTE *bitmap, int size);
    BOOL m_asyncrun;
    BOOL m_asyncstopped;
    CWinThread *m_resetthread;
    BOOL m_resetrun;
    BOOL m_resetstopped;
    CWinThread *m_isothread;
    CEvent *m_isoevent;
    CWinThread *m_asyncthread;
    CEvent *m_asyncevent;
    int NumberOfCameras();
    BOOL StopCapture();
    BOOL StartCapture();
    CPhysical();
    virtual -CPhysical();

private:
    BOOL LockWait(int camera, DWORD mask, DWORD ored, int pause, DWORD maskwait,
DWORD *value);
    BOOL ReleaseDMA();
    BOOL ReserveDMA();
    BOOL DeAllocateBandwidth();
    BOOL AllocateBandwidth();
    BOOL DeAllocateChannel();
    BOOL AllocateChannel();
    BOOL ResetFIFO();
    BOOL WaitForComplete(int camera, UINT mask, DWORD *value);
    BOOL SendMicroPacket(int camera, union micromess *out, union micromess *in);
    DWORD *m_cambuf;
    DWORD m_hostcount;
    DWORD m_cameracount;
    BOOL UnLockISOChannel();
    BOOL LockISOChannel();
    BOOL AsyncWriteBlock(int camera,unsigned int address, int size, DWORD *data);
    BOOL AsyncReadBlock(int camera,unsigned int address, int size, DWORD *data);
    BOOL AsyncLock(int camera, unsigned int address, DWORD value1, DWORD value2,
DWORD *result);
    BOOL AsyncWrite(int camera, unsigned int address, DWORD value);
    BOOL AsyncRead(int camera,unsigned int address,DWORD *value);
    CDWordArray m_handles;
};

#endif //
!defined(AFX_PHYSICAL_H__0B508AD2_296A_11D2_A11E_0000C0059AB9__INCLUDED_)
```

D.5.11.2 Physical.cpp

```
// Physical.cpp: implementation of the CPhysical class.
//
// Implementation of all 1394 commands and functions
//
// NOTE: this code only assumes one host adapter; there could be many.
//
```

```
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Larch.h"
#include "Physical.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

const int AUTORESET_EVENT   = 0;
const int MANUALRESET_EVENT = 1;
const int NONSIGNAL= 0;
const int SIGNAL    = 1;

/* Swap words for Intel LSB to Motorola MSB */
#define SWAPWORD(x) MAKEWORD(HIBYTE(x), LOBYTE(x))
#define SWAPLONG(x) MAKELONG(SWAPWORD(HIWORD(x)), SWAPWORD(LOWORD(x)))

// thread to monitor 1394 events
UINT Monitor1394Events(LPVOID lpVoid);
UINT Monitor1394IsoEvents(LPVOID lpVoid);

// DSP memory location for async command to command buffer
#define CAMERACOMMAND 0x10600

/* Set buffer sizes */
#define BUFFERPACKET 512
#define BUFFERTOTAL  65536*2

// Number of maximum retries
#define RETRY1394 2
// Timeout for retries
#define TIMEOUT1394 100

// Use 100Mbps
#define ISOSPEED S100


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CPhysical::CPhysical()
{
    m_handles.SetSize(1,1);
    m_resetrun = TRUE;
    m_resetstopped = FALSE;
    m_resetthread = NULL;

    BYTE number;

    /* Create a 1394 bus reset thread if a card exists */
    if (GetPAPISupportInfo(&number)==PAPI_NOERROR)
    {
        m_resetthread = AfxBeginThread(Monitor1394Events, this);
    }
```

```
    else
    {
        m_resetrun = FALSE;
        m_resetstopped = TRUE;
    }

    /* Some variables not completely implemented */
    m_cameracount=0;
    m_hostcount=0;

}

CPhysical::~CPhysical()
{
    // Destroy camera list
    m_handles.RemoveAll();

    // Wait for reset thread to stop
    m_resetrun = FALSE;
    while(m_resetstopped == FALSE)
    {
        Sleep(0);
    }

    delete m_cambuf;
}

/* Becareful modifying the 1394 transaction routines.  The kernel can */
/* crash if these operations are not done properly. */

/* Send an asynchronous read command to the camera */
BOOL CPhysical::AsyncRead(int camera, unsigned int address, DWORD * value)
{
    int r=0;

    /* Make sure the camera exists */
    ASSERT( camera < m_handles.GetSize() );

    /* Create command structure */
    PRB_EXEC_ASYNC PrbIo;
    PPRB_EXEC_ASYNC pPrbIo = &PrbIo;
    ASSERT( pPrbIo );

    while (r<RETRY1394)
    {
        /* Setup structure */
        memset(pPrbIo, 0, sizeof(PRB_EXEC_ASYNC));

        DWORD data;

        pPrbIo->PRB_SBC_Header.Cmd = P_EXEC_ASYNC_CMD;
        pPrbIo->PRB_SBC_Header.DevHandle = m_handles.GetAt(camera);
        pPrbIo->PRB_SBC_Header.HaNum= 0; // fix this

        /* Create an event */
        HANDLE hEvent = CreateEvent(NULL, AUTORESET_EVENT, NONSIGNAL, NULL);
        ASSERT(hEvent);

        /* addresses are set to the upper memory: 0xFFFFFxxxxxxx */
```

```
        pPrbIo->PRB_SBC_Header.OverlappedInfo.hEvent = hEvent;
        pPrbIo->Mode = PM_ASYNC_READ_REQUEST;
        pPrbIo->TargetDeviceAddressHi = 0xFFFF;
        pPrbIo->TargetDeviceAddressLo = 0xF0000000 | address;
        pPrbIo->HA_Buf_Ptr = (DWORD)&data;
        pPrbIo->TotalXfrLen = sizeof(DWORD);

        /* setup abort structure */
        PRB_TOABORT abort;
        memset(&abort, 0, sizeof(PRB_TOABORT));
        abort.PRB_SBC_Header.Cmd          = P_ABORT_1394_CMD;
        abort.PRB_SBC_Header.DevHandle    = m_handles.GetAt(camera);
        abort.PRB_SBC_Header.HaNum      = 0; // fix this
        abort.PRB_ToAbort                 = pPrbIo;

        // Wait until the data returned
        if ( SendPAPICommand(pPrbIo) == PAPI_ERROR )
        {
            /* Error occured, send abort command to kernel */
            if ( SendPAPICommand(&abort) != PS_COMP )
            {
                TRACE0("Async read failed in USER!  Leaving...\n");
                CloseHandle(hEvent);
                return FALSE;
            }
            CloseHandle(hEvent);
            TRACE0("Async read failed in USER!\n");
        }
        else
        {
            DWORD dwStatus;
            dwStatus = WaitForSingleObject(hEvent, TIMEOUT1394);

            if ( WAIT_OBJECT_0 != dwStatus )
            {
                /* Error occured, send abort command to kernel */
                if ( SendPAPICommand(&abort) != PS_COMP )
                {
                    TRACE0("Async read failed in KERNEL!  Could not Abort -
Leaving...\n");
                    CloseHandle(hEvent);
                    return FALSE;
                }
                // Close event and buffer but this is very dangerous.
                CloseHandle(hEvent);
                TRACE0("Async read failed in KERNEL!\n");
            }
            else
            {
                CloseHandle(hEvent);

                /* swap byte order since this is an intel processor */
                *value = SWAPLONG(data);

                if (r) TRACE0("Async read recovered.\n");
                return TRUE;
            }
        }
        r++;
```

```
    }

    TRACE0("Async read returned fail.\n");
    return FALSE;
}

/* Send an asynchronous lock swap command to the camera */
BOOL CPhysical::AsyncLock(int camera, unsigned int address, DWORD value1, DWORD
value2, DWORD *result)
{
    int r=0;

    /* Make sure the camera exists */
    ASSERT( camera < m_handles.GetSize() );

    /* Create command structure */
    PRB_EXEC_ASYNC PrbIo;
    PPRB_EXEC_ASYNC pPrbIo = &PrbIo;
    ASSERT( pPrbIo );

    while (r<RETRY1394)
    {
        /* Setup structure */
        memset(pPrbIo, 0, sizeof(PRB_EXEC_ASYNC));

        pPrbIo->PRB_SBC_Header.Cmd          = P_EXEC_ASYNC_CMD;
        pPrbIo->PRB_SBC_Header.DevHandle    = m_handles.GetAt(camera);
        pPrbIo->PRB_SBC_Header.HaNum= 0; // fix this

        /* Create an event */
        HANDLE hEvent = CreateEvent(NULL, AUTORESET_EVENT, NONSIGNAL, NULL);
        ASSERT(hEvent);

        /* addresses are set to the upper memory: 0xFFFFFxxxxxxx */
        pPrbIo->PRB_SBC_Header.OverlappedInfo.hEvent = hEvent;
        pPrbIo->Mode            = PM_ASYNC_LOCK_REQUEST;
        pPrbIo->Extend_tCode    = PE_MASK_SWAP; // PE_WRAP_ADD; // PE_BOUNDED_ADD;
// PE_FETCH_ADD; //PE_MASK_SWAP; // PE_COMPARE_SWAP;
        pPrbIo->TargetDeviceAddressHi = 0xFFFF;
        pPrbIo->TargetDeviceAddressLo = 0xF0000000 | address;

        // New value
        DWORD dwHaBuf[2];
        DWORD* pdwHaBuf = dwHaBuf;
        ASSERT( pdwHaBuf );

        pdwHaBuf[0] = SWAPLONG(value1);
        pdwHaBuf[1] = SWAPLONG(value2);

        pPrbIo->HA_Buf_Ptr = (DWORD)pdwHaBuf;
        pPrbIo->TotalXfrLen = sizeof(DWORD)*2;

        /* setup abort structure */
        PRB_TOABORT abort;
        memset(&abort, 0, sizeof(PRB_TOABORT));
        abort.PRB_SBC_Header.Cmd          = P_ABORT_1394_CMD;
        abort.PRB_SBC_Header.DevHandle    = m_handles.GetAt(camera);
        abort.PRB_SBC_Header.HaNum      = 0; // fix this
        abort.PRB_ToAbort                 = pPrbIo;
```

```
        // Wait until the data returned
        if ( SendPAPICommand(pPrbIo) == PAPI_ERROR )
        {
            /* Error occured, send abort command to kernel */
            if ( SendPAPICommand(&abort) != PS_COMP )
            {
                TRACE0("Async lock failed in USER!  Leaving...\n");
                CloseHandle(hEvent);
                return FALSE;
            }
            CloseHandle(hEvent);
            TRACE0("Async lock command failed in USER!\n");
        }
        else
        {
            DWORD dwStatus;
            dwStatus = WaitForSingleObject(hEvent, TIMEOUT1394);

            if ( WAIT_OBJECT_0 != dwStatus )
            {
                /* Error occured, send abort command to kernel */
                if ( SendPAPICommand(&abort) != PS_COMP )
                {
                    TRACE0("Async lock failed in KERNEL!  Could not Abort -
Leaving...\n");
                    CloseHandle(hEvent);
                    return FALSE;
                }
                // Close event and buffer but this is very dangerous.
                CloseHandle(hEvent);
                TRACE0("Async lock failed in KERNEL!\n");
            }
            else
            {
                CloseHandle(hEvent);

                if (result) *result = SWAPLONG(pdwHaBuf[0]);

                if (r) TRACE0("Async lock recovered.\n");
                return TRUE;
            }
        }
        r++;
    }

    TRACE0("Async lock returned fail.\n");
    return FALSE;
}

/* Send an asynchronous write command to the camera */
BOOL CPhysical::AsyncWrite(int camera, unsigned int address, DWORD value)
{
    int r=0;

    /* Make sure the camera exists */
    ASSERT( camera < m_handles.GetSize() );

    /* Create command structure */
```

```
PRB_EXEC_ASYNC PrbIo;
PPRB_EXEC_ASYNC pPrbIo = &PrbIo;
ASSERT( pPrbIo );

while (r<RETRY1394)
{
    /* Setup structure */
    memset(pPrbIo, 0, sizeof(PRB_EXEC_ASYNC));

    DWORD data = SWAPLONG(value);

    pPrbIo->PRB_SBC_Header.Cmd = P_EXEC_ASYNC_CMD;
    pPrbIo->PRB_SBC_Header.DevHandle = m_handles.GetAt(camera);
    pPrbIo->PRB_SBC_Header.HaNum= 0; // fix this

    /* Create an event */
    HANDLE hEvent = CreateEvent(NULL, AUTORESET_EVENT, NONSIGNAL, NULL);
    ASSERT(hEvent);

    pPrbIo->PRB_SBC_Header.OverlappedInfo.hEvent = hEvent;

    /* addresses are set to the upper memory: 0xFFFFFxxxxxxx */
    pPrbIo->Mode = PM_ASYNC_WRITE_REQUEST;
    pPrbIo->TargetDeviceAddressHi = 0xFFFF;
    pPrbIo->TargetDeviceAddressLo = 0xF0000000 | address;
    pPrbIo->HA_Buf_Ptr = (DWORD)&data;
    pPrbIo->TotalXfrLen = sizeof(DWORD);

    /* setup abort structure */
    PRB_TOABORT abort;
    memset(&abort, 0, sizeof(PRB_TOABORT));
    abort.PRB_SBC_Header.Cmd          = P_ABORT_1394_CMD;
    abort.PRB_SBC_Header.DevHandle    = m_handles.GetAt(camera);
    abort.PRB_SBC_Header.HaNum      = 0; // fix this
    abort.PRB_ToAbort                 = pPrbIo;

    // Wait until the data returned
    if ( SendPAPICommand(pPrbIo) == PAPI_ERROR )
    {
        /* Error occured, send abort command to kernel */
        if ( SendPAPICommand(&abort) != PS_COMP )
        {
            TRACE0("Async write failed in USER!  Leaving...\n");
            CloseHandle(hEvent);
            return FALSE;
        }
        CloseHandle(hEvent);
        TRACE0("Async write failed in USER!\n");
    }
    else
    {
        // Wait for the result to come back within certain amount of time
        DWORD dwStatus;
        dwStatus = WaitForSingleObject(hEvent, TIMEOUT1394);

        if ( WAIT_OBJECT_0 != dwStatus )
        {
            /* Error occured, send abort command to kernel */
            if ( SendPAPICommand(&abort) != PS_COMP )
```

```
            {
                TRACE0("Async write failed in KERNEL!  Could not Abort -
Leaving...\n");
                CloseHandle(hEvent);
                return FALSE;
            }
            // Close event and buffer but this is very dangerous.
            CloseHandle(hEvent);
            TRACE0("Async write failed in KERNEL!\n");
        }
        else
        {
            CloseHandle(hEvent);

            return TRUE;
        }
    }
    r++;
  }

  TRACE0("Async write returned fail.\n");
  return FALSE;
}

/* Send an asynchronous read block command to the camera */
BOOL CPhysical::AsyncReadBlock(int camera, unsigned int address, int size, DWORD
* data)
{
  /* Make sure the camera exists */
  ASSERT( camera < m_handles.GetSize() );

  /* Create command structure */
  PRB_EXEC_ASYNC PrbIo;
  PPRB_EXEC_ASYNC pPrbIo = &PrbIo;
  ASSERT( pPrbIo );

  DWORD temp[128];

  /* Setup structure */
  memset(pPrbIo, 0, sizeof(PRB_EXEC_ASYNC));

  pPrbIo->PRB_SBC_Header.Cmd = P_EXEC_ASYNC_CMD;
  pPrbIo->PRB_SBC_Header.DevHandle = m_handles.GetAt(camera);
  pPrbIo->PRB_SBC_Header.HaNum= 0; // fix this

  /* Create an event */
  HANDLE hEvent = CreateEvent(NULL, AUTORESET_EVENT, NONSIGNAL, NULL);
  ASSERT(hEvent);

  /* addresses are set to the upper memory: 0xFFFFFxxxxxxx */
  pPrbIo->PRB_SBC_Header.OverlappedInfo.hEvent = hEvent;
  pPrbIo->Mode = PM_ASYNC_READ_REQUEST;
  pPrbIo->TargetDeviceAddressHi = 0xFFFF;
  pPrbIo->TargetDeviceAddressLo = 0xF0000000 | address;
  pPrbIo->TotalXfrLen = sizeof(DWORD)*128;
  pPrbIo->HA_Buf_Ptr = (DWORD)temp;

  /* setup abort structure */
  PRB_TOABORT abort;
```

```
    memset(&abort, 0, sizeof(PRB_TOABORT));
    abort.PRB_SBC_Header.Cmd          = P_ABORT_1394_CMD;
    abort.PRB_SBC_Header.DevHandle    = m_handles.GetAt(camera);
    abort.PRB_SBC_Header.HaNum      = 0; // fix this
    abort.PRB_ToAbort                 = pPrbIo;

    // 128 quads in each block
    int blocks = size >> 7;
    int rem = size & 127;
    unsigned int j;

    int i;
    for(i=0;i<blocks+(rem!=0);i++)
    {
        if (i==blocks) pPrbIo->TotalXfrLen = sizeof(DWORD)*rem;

        // Wait until the data returned
        if ( SendPAPICommand(pPrbIo) == PAPI_ERROR )
        {
            /* Error occured, send abort command to kernel */
            if ( SendPAPICommand(&abort) != PS_COMP )
            {
                TRACE0("Async read block failed in USER!  Leaving...\n");
                CloseHandle(hEvent);
                return FALSE;
            }
            CloseHandle(hEvent);
            TRACE0("Async read block failed in USER!  Leaving...\n");
            return FALSE;
        }
        else
        {
            DWORD dwStatus;
            dwStatus = WaitForSingleObject(hEvent, TIMEOUT1394);

            if ( WAIT_OBJECT_0 != dwStatus )
            {
                /* Error occured, send abort command to kernel */
                if ( SendPAPICommand(&abort) != PS_COMP )
                {
                    TRACE0("Async read block failed in KERNEL!  Could not Abort -
Leaving...\n");
                    CloseHandle(hEvent);
                    return FALSE;
                }
                // Close event and buffer but this is very dangerous.
                CloseHandle(hEvent);
                TRACE0("Async read block failed in KERNEL!  Leaving...\n");
                return FALSE;
            }
            else
            {
                for (j=0;j<(pPrbIo->TotalXfrLen/sizeof(DWORD));j++)
                {
                    data[j] = SWAPLONG(temp[j]);
                }

                pPrbIo->TargetDeviceAddressLo += sizeof(DWORD)*128;
                data += 128;
```

```
            }
        }
    }

    CloseHandle(hEvent);

    return TRUE;
}

/* Send an asynchronous write block command to the camera */
BOOL CPhysical::AsyncWriteBlock(int camera, unsigned int address, int size,
DWORD * data)
{
    /* Make sure the camera exists */
    ASSERT( camera < m_handles.GetSize() );

    /* Create command structure */
    PRB_EXEC_ASYNC PrbIo;
    PPRB_EXEC_ASYNC pPrbIo = &PrbIo;
    ASSERT( pPrbIo );
    memset(pPrbIc, 0, sizeof(PRB_EXEC_ASYNC));

/* Problems with Adaptec Card with block writes more than 4 bytes */
/*  When resolves, adjust this value. */
#define writeblocksize 4

    DWORD temp[writeblocksize];

    /* Setup structure */
    pPrbIo->PRB_SBC_Header.Cmd = P_EXEC_ASYNC_CMD;
    pPrbIo->PRB_SBC_Header.DevHandle = m_handles.GetAt(camera);
    pPrbIo->PRB_SBC_Header.HaNum= 0; // fix this

    /* Create an event */
    HANDLE hEvent = CreateEvent(NULL, AUTORESET_EVENT, NONSIGNAL, NULL);
    ASSERT(hEvent);

    pPrbIo->PRB_SBC_Header.OverlappedInfo.hEvent = hEvent;

    /* addresses are set to the upper memory: 0xFFFFFxxxxxxx */
    pPrbIo->Mode = PM_ASYNC_WRITE_REQUEST;
    pPrbIo->TargetDeviceAddressHi = 0xFFFF;
    pPrbIo->TargetDeviceAddressLo = 0xF0000000 | address;
    pPrbIo->TotalXfrLen = sizeof(DWORD)*writeblocksize;
    pPrbIo->HA_Buf_Ptr = (DWORD)temp;

    /* setup abort structure */
    PRB_TOABORT abort;
    memset(&abort, 0, sizeof(PRB_TOABORT));
    abort.PRB_SBC_Header.Cmd          = P_ABORT_1394_CMD;
    abort.PRB_SBC_Header.DevHandle    = m_handles.GetAt(camera);
    abort.PRB_SBC_Header.HaNum      = 0; // fix this
    abort.PRB_ToAbort                 = pPrbIo;

    int blocks = size / writeblocksize;
    int rem = size % writeblocksize;
    unsigned int j;

    int i;
```

```
    for(i=0;i<blocks+(rem!=0);i++)
    {
        if (i==blocks) pPrbIo->TotalXfrLen = sizeof(DWORD)*rem;

        for (j=0;j<(pPrbIo->TotalXfrLen/sizeof(DWORD));j++)
        {
            temp[j] = SWAPLONG(data[j]);
        }

        // Wait until the data returned
        if ( SendPAPICommand(pPrbIo) == PAPI_ERROR )
        {
            /* Error occured, send abort command to kernel */
            if ( SendPAPICommand(&abort) != PS_COMP )
            {
                TRACE0("Async write block failed in USER!  Leaving...\n");
                CloseHandle(hEvent);
                return FALSE;
            }
            CloseHandle(hEvent);
            TRACE0("Async write block failed in USER!  Leaving...\n");
            return FALSE;
        }
        else
        {
            // Wait for the result to come back within certain amount of time
            DWORD dwStatus;
            dwStatus = WaitForSingleObject(hEvent, TIMEOUT1394);

            if ( WAIT_OBJECT_0 != dwStatus )
            {
                /* Error occured, send abort command to kernel */
                if ( SendPAPICommand(&abort) != PS_COMP )
                {
                    TRACE0("Async write block failed in KERNEL!  Could not Abort -
Leaving...\n");
                    CloseHandle(hEvent);
                    return FALSE;
                }
                // Close event and buffer but this is very dangerous.
                CloseHandle(hEvent);
                TRACE0("Async write block failed in KERNEL!  Leaving...\n");
                return FALSE;
            }
            else
            {
                pPrbIo->TargetDeviceAddressLo += sizeof(DWORD)*writeblocksize;
                data += writeblocksize;
            }
        }
    }

    CloseHandle(hEvent);

    return TRUE;
}

/* Check the bus for all the cameras on the 1394 BUS */
int CPhysical::NumberOfCameras()
```

```
{
    BYTE number=0;

    /* Make sure we have a card */
    if (GetPAPISupportInfo(&number)==PAPI_NOERROR)
    {
        m_handles.RemoveAll();

        /* Quick hack for testing the single camera */
        /* After a bus reset, the Adaptec function to read the cameras CSR ROM */
        /*  Directory is VERY SLOW (I can do it faster), they have yet to resolve
*/
        /*  this problem...  My guess is that they all task changing every time */
        /*  a quadlet is received.  I have replicated the function myself and it
*/
        /*  is much faster than their routine. */
        /* We simple place the ID of our camera (0x0000d101 0x00000001) which can
*/
        /*  be found in the DSP code. */

        DWORD   dwNodeUniqueIdHi, dwNodeUniqueIdLo;

        dwNodeUniqueIdHi=0x0000d101;
        dwNodeUniqueIdLo=0x00000001;

        SBC_GET_HANDLE  sbcGetHandle;

        memset(&sbcGetHandle, 0, sizeof(SBC_GET_HANDLE));
        sbcGetHandle.PRB_SBC_Header.Cmd     = P_GET_HANDLE;
        sbcGetHandle.PRB_SBC_Header.HaNum   = 0; // Adapter ID
        sbcGetHandle.NodeUniqueID_HI        = dwNodeUniqueIdHi;
        sbcGetHandle.NodeUniqueID_LO        = dwNodeUniqueIdLo;

        if ( (BusConfig(&sbcGetHandle) != PAPI_NOERROR) ||
            (PS_COMP != sbcGetHandle.PRB_SBC_Header.Status) )
        {
            TRACE0("Could't get camera.\n");
            return 0;
        }

        /* Add hanle to camera to list */
        m_handles.Add(sbcGetHandle.PRB_SBC_Header.DevHandle);

        // Bench mark async test at 100Mbps; 1.5MB/s in slave mode
/*
        time_t time1,time2;

        DWORD *test = new DWORD[128*65];
        ASSERT(test);
        int a,i;
        int j;

        time(&time1);

        for(j=0;j<1;j++) // 100
        for(i=0;i<1;i++) // 32
        {
            a = AsyncReadBlock(0,0x20000,128*64,test);
            if (a)
```

```
            {
                AfxMessageBox("Failed!");
            }
        }
        delete test;

        time(&time2);

        a=time2-time1;
*/

        AsyncWrite(0,CAMERACOMMAND+0x04,0xf002ffc1);
        AsyncWrite(0,CAMERACOMMAND+0x08,0xf0010000);

        union micromess out,in;

        /* Get camera name */
        out.form.length = 0;
        out.form.type = 6;
        SendMicroPacket(0,&out,&in);
        TRACE1("Camera Device is : %s\n",in.form.data);

        /* Get camera version */
        out.form.length = 0;
        out.form.type = 5;
        SendMicroPacket(0,&out,&in);
        in.form.data[4] = 0;
        TRACE1("Camera Version is: %s\n",in.form.data);

        /* Get camera serial number */
        out.form.length = 0;
        out.form.type = 8;
        SendMicroPacket(0,&out,&in);
        TRACE1("Camera Serial is : %s\n",in.form.data);

        /* Set camera parameters */
        out.form.length = 6;
        out.form.type = 0x32;
        out.form.data[0] = 5;
        out.form.data[1] = 16;
        out.form.data[2] = 0;
        out.form.data[3] = 1;
        out.form.data[4] = 2;
        out.form.data[5] = 3;
        SendMicroPacket(0,&out,&in);

        /* Set camera speed parameters */
        out.form.length = 5;
        out.form.type = 0x33;
        out.form.data[0] = 2;
        out.form.data[1] = 4;
        out.form.data[2] = 6; // 4 5 6 7
        out.form.data[3] = 16;
        out.form.data[4] = in.form.data[1] & 0xf3 | 0x08; // 0 4 8 c
//      out.form.data[4] = in.form.data[1] & 0xcf | 0x20;
        SendMicroPacket(0,&out,&in);


        /* Send other non-documented camera commands, change TDI stages, etc. */
```

```
/*
        out.form.length = 2;
        out.form.type = 0x32;
        out.form.data[0] = 1;
        out.form.data[1] = 12;
        SendMicroPacket(0,&out,&in);
        TRACE1("Parameter: %d\n",in.form.data[1]);
*/

        return 1;
    }
    else
    {
        return -1;
    }
}

/* New thread to monitor bus resets */
UINT Monitor1394Events(LPVOID lpVoid)
{

    // working variables
    HANDLE          brEventHandle;
    PRB_BUSEVENTbusEvent;
    CPhysical  *mData;
    DWORD           dwStatus;

    // get monitor data
    mData = (CPhysical*)lpVoid;

    // get a pointer to the parent object
    CWnd *pParent = (CWnd*)theApp.m_pMainWnd;
    if (! pParent) return 0;

    // create an event
    brEventHandle = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (! brEventHandle) return 0;

    mData->m_resetstopped = FALSE;

    // loop for-ever
    while (mData->m_resetrun)
    {

        // initialization
        ZeroMemory(&busEvent, sizeof(PRB_BUSEVENT));

        // setup papi commands
        busEvent.PRB_SBC_Header.Cmd= P_GET_BUS_EVENT;
        busEvent.PRB_SBC_Header.HaNum= 0; // fix this for multiple adapters
        busEvent.PRB_SBC_Header.OverlappedInfo.hEvent   = brEventHandle;
        busEvent.BusEventBitMap  = BE_BUS_RESET;

        // monitor event, if it fails perhaps ad retry code later (or will generic
monitor thread handle it ?)
        if (SendPAPICommand(&busEvent) != PAPI_NOERROR)
            break;

        // switch based on status
```

```
        switch (busEvent.PRB_SBC_Header.Status)
        {

            case PS_PENDING:

                // pre-set for proper loop function
                dwStatus = WAIT_TIMEOUT;

                // keep looping until we are asked to terminate of the event
occurs.
                while ((dwStatus == WAIT_TIMEOUT) && (mData->m_resetrun))
                {

                    // wait for the event
                    dwStatus = WaitForSingleObject(brEventHandle, 1000);

                    // event is here
                    if (WAIT_OBJECT_0 == dwStatus)
                        pParent->PostMessage(WM_1394RESET);

                } // end while

            break;

            case PS_COMP:

                /* Send custom message to application message queue */
                pParent->PostMessage(WM_1394RESET);

            break;

        } // end switch

    } // end while

    // close the handle
    CloseHandle(brEventHandle);

    mData->m_resetstopped = TRUE;
    TRACE0("ResetThread shutdown by request.\n");

    // exit thread
    AfxEndThread(0);
    return 0;

}

/* Another thread to monitor 1394 isochronous complete transactions */
UINT Monitor1394IsoEvents(LPVOID lpVoid)
{
    int tempcount = 0;
    int z;

#define NumBuffers 16

    // working variables
    HANDLE        isoEventHandle[NumBuffers];
    CPhysical  *mData;
    DWORD         dwStatus;
```

```
    // get monitor data
    mData = (CPhysical*)lpVoid;

    // get a pointer to the parent object
    CWnd *pParent = (CWnd*)theApp.m_pMainWnd;
    if (! pParent) return 0;

    mData->m_asyncstopped = FALSE;

    DWORD test=0;
    DWORD received=0;
    BYTE *indata;

    /* Allocate buffer pointers */
    DWORD *buffer[NumBuffers];

    // create an event for each buffer
    for (z=0;z<NumBuffers;z++)
    {
        isoEventHandle[z] = CreateEvent(NULL, TRUE, FALSE, NULL);
        if (!isoEventHandle[z]) return 0;
        buffer[z] = new DWORD[BUFFERTOTAL/4];
        if (!buffer[z]) return 0;
    }

    PRB_QUEUE_ISOCisoEvent[NumBuffers];

    /* send iso queue command for each buffer */
    for (z=0;z<NumBuffers;z++)
    {
        // initialization
        ZeroMemory(&isoEvent[z], sizeof(PRB_QUEUE_ISOC));

        // setup papi commands
        isoEvent[z].PRB_SBC_Header.Cmd= P_QUEUE_ISOC_CMD;
        isoEvent[z].PRB_SBC_Header.HaNum= 0; // fix this for multiple adapters
        isoEvent[z].PRB_SBC_Header.OverlappedInfo.hEvent    = isoEventHandle[z];
        isoEvent[z].Mode = PM_ISOC_READ;
        isoEvent[z].Channel = 0;
        isoEvent[z].Speed = ISOSPEED;
        isoEvent[z].SyncCode = 0;
        isoEvent[z].CycleStart = 0; // don't know
        isoEvent[z].PrimaryXmitPayload = BUFFERPACKET;
        isoEvent[z].SecondaryXmitPayload = 0;
        isoEvent[z].HA_Buf_Ptr = (DWORD)buffer[z];
        isoEvent[z].TotalXfrLen = BUFFERTOTAL;

        dwStatus = SendPAPICommand(&isoEvent[z]);
        if ((PAPI_NOERROR != dwStatus) || (PS_ERROR ==
isoEvent[z].PRB_SBC_Header.Status))
        {
            /* Leave if we couldn't complete the command */
            TRACE0("Failure on 0!\n");
            goto leave;
        }
    }

    /* Start isochronous receiving */
```

```
    PRB_START_ISOC IsocXtrl;
    memset(&IsocXtrl, 0, sizeof(PRB_START_ISOC));

    IsocXtrl.PRB_SBC_Header.Cmd   = P_START_ISOC;
    IsocXtrl.PRB_SBC_Header.HaNum = 0;  // fix this
    IsocXtrl.Channel              = 0;

    dwStatus = SendPAPICommand(&IsocXtrl);
    if (PAPI_NOERROR != dwStatus)
    {
       /* Leave if we couldn't complete the command */
       TRACE0("Failure on 1!\n");
       goto leave;
    }

    // loop for-ever until parent process says so
    while (mData->m_asyncrun)
    {
       /* Cycle through each buffer to receive isochronous data */
       for (z=0;z<NumBuffers;z++)
       {
          dwStatus = WAIT_TIMEOUT;
          while ( (dwStatus == WAIT_TIMEOUT) && (mData->m_asyncrun) )
          {
             // wait for the event
             dwStatus = WaitForSingleObject(isoEventHandle[z], 1000);

             // event is here
             if (WAIT_OBJECT_0 == dwStatus)
             {
                indata = (BYTE*)buffer[z];

                /* Bytes received will be what we asked for */
                received = BUFFERTOTAL;

                /* Send a message to parent thread to process the data */
                pParent-
>PostMessage(WM_CAMERASIG1,(LPARAM)indata,(LPARAM)received);
             }
          }

          /* Repost isochronous queue command */
          if (mData->m_asyncrun)
          {
             // initialization
             ZeroMemory(&isoEvent[z], sizeof(PRB_QUEUE_ISOC));

             // setup papi commands
             isoEvent[z].PRB_SBC_Header.Cmd= P_QUEUE_ISOC_CMD;
             isoEvent[z].PRB_SBC_Header.HaNum= 0; // fix this for multiple
adapters
             isoEvent[z].PRB_SBC_Header.OverlappedInfo.hEvent   =
isoEventHandle[z];
             isoEvent[z].Mode = PM_ISOC_READ;
             isoEvent[z].Channel = 0;
             isoEvent[z].Speed = ISOSPEED;
             isoEvent[z].SyncCode = 0;
             isoEvent[z].CycleStart = 0; // don't know
             isoEvent[z].PrimaryXmitPayload = BUFFERPACKET;
```

```
            isoEvent[z].SecondaryXmitPayload = 0;
            isoEvent[z].IsocIoRequestFlags = 0;
            isoEvent[z].HA_Buf_Ptr = (DWORD)buffer[z];
            isoEvent[z].TotalXfrLen = BUFFERTOTAL;

            // reset event
            ResetEvent(isoEventHandle[z]);

            dwStatus = SendPAPICommand(&isoEvent[z]);
            if ((PAPI_NOERROR != dwStatus) || (PS_ERROR ==
isoEvent[z].PRB_SBC_Header.Status))
            {
                /* Leave if we couldn't complete the command */
                TRACE0("Failure on 3!\n");
                goto leave;
            }
        }
        /* Exit the while loop when asked to */
        if (!mData->m_asyncrun)
        {
            z=NumBuffers+5;;
        }
    }
} // end while

/* Abort all isochronous queued commands */
PRB_TOABORT abort;
for (z=NumBuffers-1;z>=0;z--)
{
    memset(&abort, 0, sizeof(PRB_TOABORT));
    abort.PRB_SBC_Header.Cmd          = P_ABORT_1394_CMD;
    abort.PRB_SBC_Header.DevHandle    = 1;
    abort.PRB_SBC_Header.HaNum      = 0; // fix this
    abort.PRB_ToAbort                 = &isoEvent[z];

    // Wait until the data returned
    if ( SendPAPICommand(&abort) != PS_COMP )
    {
        TRACE0("Failed to abort iso command.\n");
    }
}

TRACE0("  Iso command aborted.\n");
Sleep(500);

leave:

    /* Stop isochronous receiving */
    memset(&IsocXtrl, 0, sizeof(PRB_STOP_ISOC));

    IsocXtrl.PRB_SBC_Header.Cmd   = P_STOP_ISOC;
    IsocXtrl.PRB_SBC_Header.HaNum = 0;  // fix this
    IsocXtrl.Channel              = 0;

    dwStatus = SendPAPICommand(&IsocXtrl);
    if (PAPI_NOERROR != dwStatus)
    {
        /* This should never happen, if it does, the system will probably */
        /* be unstable */
```

```
        TRACE0("Failure on last!\n");
    }
    TRACE0("  Iso stopped.\n");
    Sleep(500);

    // close the handle
    for (z=NumBuffers-1;z>=0;z--)
    {
        CloseHandle(isoEventHandle[z]);

        delete buffer[z];
    }

    mData->m_asyncstopped = TRUE;
    TRACE0("Iso Listen Thread shutdown by request.\n");

    // exit thread
    AfxEndThread(0);
    return 0;

}

/* Setup the ISO channel for capture */
BOOL CPhysical::LockISOChannel()
{
    TRACE0("Setting up ISO channel...\n");

    /* If we can't reserve the DMA, chances are something failed elsewhere */
    if (ReserveDMA()==FALSE)
    {
        ReleaseDMA();
        ReserveDMA();
    }
    /* These don't seem to apply in this environment, ideally, in the future */
    /*  this should have to be used by the Microsoft implementation. */
//  AllocateBandwidth();
//  AllocateChannel();

    return TRUE;
}

/* Release ISO channel for capture */
BOOL CPhysical::UnLockISOChannel()
{
    TRACE0("Shutting down ISO channel...\n");

    /* Again, these don't apply */
//  DeAllocateChannel();
//  DeAllocateBandwidth();
    ReleaseDMA();

    return TRUE;
}

/* Allocate Channel */
BOOL CPhysical::AllocateChannel()
{
    SBC_CHANNEL Sbc;
    PSBC_CHANNEL pSbc = &Sbc;
```

```
    ASSERT( pSbc );

    /* Set up structure */
    memset(pSbc, 0, sizeof(SBC_CHANNEL));
    pSbc->PRB_SBC_Header.Cmd= P_ALLOCATE_CHANNEL;
    pSbc->PRB_SBC_Header.HaNum= 0; // fix this
    pSbc->ChannelRequested = 0;
    pSbc->ChannelAllocated = 0;

    /* Send command */
    BusConfig(pSbc);

    if ( PS_COMP != pSbc->PRB_SBC_Header.Status )
    {
        TRACE0("ALLOCATE_CHANNEL failed.\n");
        return FALSE;
    }

    return TRUE;
}

/* DeAllocate Channel */
BOOL CPhysical::DeAllocateChannel()
{
    SBC_CHANNEL Sbc;
    PSBC_CHANNEL pSbc = &Sbc;
    ASSERT( pSbc );

    /* Set up structure */
    memset(pSbc, 0, sizeof(SBC_CHANNEL));
    pSbc->PRB_SBC_Header.Cmd= P_DEALLOCATE_CHANNEL;
    pSbc->PRB_SBC_Header.HaNum= 0; // fix this
    pSbc->ChannelRequested = 0;
    pSbc->ChannelAllocated = 0;

    /* Send command */
    BusConfig(pSbc);

    if ( PS_COMP != pSbc->PRB_SBC_Header.Status )
    {
        TRACE0("DEALLOCATE_CHANNEL failed.\n");
        return FALSE;
    }

    return TRUE;
}

/* Allocate Bandwidth */
BOOL CPhysical::AllocateBandwidth()
{
    SBC_BANDWIDTH Sbc;
    PSBC_BANDWIDTH pSbc = &Sbc;
    ASSERT( pSbc );

    /* Set up structure */
    memset(pSbc, 0, sizeof(SBC_BANDWIDTH));
    pSbc->PRB_SBC_Header.Cmd= P_ALLOCATE_BANDWIDTH;
    pSbc->PRB_SBC_Header.HaNum= 0; // fix this
    pSbc->Speed = 0; // S100
```

```
    pSbc->BandwidthRequested = (4+3)*16 + 32;
    pSbc->BandwidthAvailable = 0;

    /* Send command */
    BusConfig(pSbc);

    if ( PS_COMP != pSbc->PRB_SBC_Header.Status )
    {
        TRACE0("ALLOCATE_BANDWIDTH failed.\n");
        return FALSE;
    }

    return TRUE;
}

/* DeAllocate Bandwidth */
BOOL CPhysical::DeAllocateBandwidth()
{
    SBC_BANDWIDTH Sbc;
    PSBC_BANDWIDTH pSbc = &Sbc;
    ASSERT( pSbc );

    /* Set up structure */
    memset(pSbc, 0, sizeof(SBC_BANDWIDTH));
    pSbc->PRB_SBC_Header.Cmd= P_DEALLOCATE_BANDWIDTH;
    pSbc->PRB_SBC_Header.HaNum= 0; // fix this
    pSbc->Speed = 0; // S100
     pSbc->BandwidthRequested = (4+3)*16 + 32;
     pSbc->BandwidthAvailable = 0;

    /* Send command */
    BusConfig(pSbc);

    if ( PS_COMP != pSbc->PRB_SBC_Header.Status )
    {
        TRACE0("DEALLOCATE_BANDWIDTH failed.\n");
        return FALSE;
    }

    return TRUE;
}

/* Reserve DMA for ISO transmission */
BOOL CPhysical::ReserveDMA()
{
    SBC_ISOCH_RESOURCE Sbc;
    PSBC_ISOCH_RESOURCE pSbc = &Sbc;
    ASSERT( pSbc );

    /* Set up structure */
    memset(pSbc, 0, sizeof(SBC_ISOCH_RESOURCE));
    pSbc->PRB_SBC_Header.Cmd= P_RESERVE_ISOC_DMA;
    pSbc->PRB_SBC_Header.HaNum= 0; // fix this
    pSbc->PRB_SBC_Header.DevHandle  = 1;
    pSbc->Channel = 0;
    pSbc->ResourceType = ISOCHRONOUS_READ;

    /* Send command */
    BusConfig(pSbc);
```

```
    if ( PS_COMP != pSbc->PRB_SBC_Header.Status )
    {
        TRACE0("RESERVE_ISOC_DMA failed.\n");
        return FALSE;
    }

    return TRUE;
}

/* Release DMA for ISO transmission */
BOOL CPhysical::ReleaseDMA()
{
    SBC_ISOCH_RESOURCE Sbc;
    PSBC_ISOCH_RESOURCE pSbc = &Sbc;
    ASSERT( pSbc );

    /* Set up structure */
    memset(pSbc, 0, sizeof(SBC_ISOCH_RESOURCE));
    pSbc->PRB_SBC_Header.Cmd= P_RELEASE_ISOC_DMA;
    pSbc->PRB_SBC_Header.HaNum= 0; // fix this
    pSbc->PRB_SBC_Header.DevHandle  = 1;
    pSbc->Channel = 0;
    pSbc->ResourceType = ISOCHRONOUS_READ;

    /* Send command */
    BusConfig(pSbc);

    if ( PS_COMP != pSbc->PRB_SBC_Header.Status )
    {
        TRACE0("RELEASE_ISOC_DMA failed.\n");
        return FALSE;
    }
    TRACE0("  DMA released.\n");
    Sleep(1000);

    return TRUE;
}

/* Send a packet to the microcontroller, and wait for a response */
BOOL CPhysical::SendMicroPacket(int camera, union micromess *out, union
micromess *in)
{
    int i,wait;

    /* Setup structure for MCU */
    out->form.length += 3;
    out->form.to = 4;
    out->form.from = 2;

    wait = (7+out->form.length+2) * 2;

    /* swap words */
    for (i=0;i<29;i++)
    {
        out->raw.data[i] = SWAPWORD(out->raw.data[i]);
    }

    /* Write them into the camera rs232 out buffer */
```

```
    AsyncWriteBlock(0,0x1073c,15,(DWORD*)out);

    /* Send command to transmit the packet */
    AsyncLock(0,CAMERACOMMAND,0x0000000c,0x00000002,NULL);

    /* Wait a while based on the length of the packet */
    Sleep(wait);

    DWORD value;

    /* wait for process to finish */
    WaitForComplete(camera,0x00000002,&value);

    /* Output status */
    TRACE1("CPhysical::SendMicroPacket = %x\n",value);
    /* should add retry on error here */

    /* Read the response packet from the MCU */
    AsyncReadBlock(0,0x10700,15,(DWORD*)in);

    /* Swap the words again */
    for (i=0;i<29;i++)
    {
        in->raw.data[i] = SWAPWORD(in->raw.data[i]);
        out->raw.data[i] = SWAPWORD(out->raw.data[i]);
    }

    /* fix this, don't always return true if there is a failure */
    return TRUE;
}

/* Send the fpga bitstream to the camera and tell it to program it */
BOOL CPhysical::LoadFPGA(BYTE *bitmap, int size)
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    int i;
    BYTE data[32768];

    /* Move bitstream into temporary buffer */
    for (i=0;i<size;i++)
        data[i]=bitmap[i];

    /* Swap the words */
    for (i=0;i<16384;i++)
    {
        ((WORD *)data)[i] = SWAPWORD(((WORD *)data)[i]);
    }

    /* Write them into the buffer for loading in the DSP */
    AsyncWriteBlock(0,0x40000,32768/4,(DWORD *)data);

    DWORD value;

    /* Send the command to program the FPGA */
    AsyncLock(0,CAMERACOMMAND,0x00000020,0x00000010,NULL);
    Sleep(500);
    WaitForComplete(camera,0x00000010,&value);

    /* Return false on an error */
```

```
    if (value & 0x00000020) return FALSE;

    return TRUE;
}

/* Reset the FIFOs by sending a series of commands to the FPGA */
BOOL CPhysical::ResetFIFO()
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    DWORD value;

    /* Assert RS lines on FIFO */
    if (LockWait(camera,0xffff0000,0x18010040,10,0x00000040,&value)==FALSE)
return FALSE;

    /* Release RS lines on FIFO */
    if (LockWait(camera,0xffff0000,0x18000040,10,0x00000040,&value)==FALSE)
return FALSE;

    /* Program almost flags */
    if (LockWait(camera,0xffff0000,0x18020040,10,0x00000040,&value)==FALSE)
return FALSE;

    /* Clear control lines */
    if (LockWait(camera,0xffff0000,0x18000040,10,0x00000040,&value)==FALSE)
return FALSE;

    return TRUE;
}

/* Prepare the 1394 card for capture */
BOOL CPhysical::StartCapture()
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    DWORD value;

    /* Setup the ISO channel, if it fails, leave! serious error */
    ASSERT(LockISOChannel());

    /* begin ISO thread */
    m_asyncrun = TRUE;
    m_asyncstopped = FALSE;
    m_asyncthread = AfxBeginThread(Monitor1394IsoEvents, this);

    /* Reset the FIFOs */
    ResetFIFO();

    /* Set capture flag */
    if (LockWait(camera,0xffff0000,0x180400c0,200,0x00000040,&value)==FALSE)
    {
        StopCapture();
        return FALSE;
    }

    /* Read back it back to make sure */
    LockWait(camera,0xffff0000,0x080400c0,200,0x00000040,&value);

    return TRUE;
}
```

```
/* Shutdown the 1394 card for capture */
BOOL CPhysical::StopCapture()
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    DWORD value;

    /* Stop the capture in the camera */
    LockWait(camera,0xffff0080,0x18000040,100,0x000000c0,&value);

    /* Stop the ISO thread, should probably use a semiphore for this */
    m_asyncrun = FALSE;
    while(m_asyncstopped == FALSE)
    {
        Sleep(0);
    }

    /* Release the ISO channel */
    ASSERT(UnLockISOChannel());

    return TRUE;
}

/* Send coded processing algorithm parameters to the camera */
BOOL CPhysical::SendCoded(BYTE *coded)
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    int i;
    DWORD value;

    // Bit 12 = 0 for read, 1 for write
    // For Write:
    //   Bit 11 = 0 for 64 bit register
    //     64 bit register (8x8)
    //        bits 10 to 8 (0 to 7) address
    //        bits 7 to 0 data
    //   Bit 11 = 1 for RAM write
    //     RAM Write
    //        bit 10 = 0 for RAM Address write/Base register write
    //          Base register write
    //            bits 9 to 8 = "00", "01", "10"
    //            bits 7 to 0 data
    //          RAM Address write
    //            bits 9 to 8 = "11"
    //            bits 7 to 0 data (address lines)
    //        bit 10 = 1 for RAM Data write
    //          RAM Data Write
    //            Low byte
    //              bits 9 to 8 = "00", "10"
    //              bits 7 to 0 data (low byte)
    //            High byte
    //              bits 9 to 8 = "01", "11"
    //              bits 7 to 0 data (high byte)
    // For Read:
    //   Bit 11 = 1 for base register input
    //     Base register input
    //       8 bits move to output shifter
    //   Bit 11 = 0 for 64 bit register read
    //     64 bit register read (8x8)
```

```
    //        bits 10 to 8 (0 to 7) address
    //        data moved to output shifter

    TRACE0("Sending Parameters:");
    /* Send all 8 */
    for(i=0;i<8;i++)
    {
        // Write
        if (LockWait(camera,0xffff0000,((0x1000 | (i << 8) | coded[i]) << 16) |
0x0040,100,0x00000040,&value)==FALSE) return FALSE;
        // Read back
//      if (LockWait(camera,0xffff0000,((0x0000 | (i << 8) | 0 ) << 16) |
0x0040,10,0x00000040,&value)==FALSE) return FALSE;
        TRACE1(" %2x",coded[i]);
    }
    TRACE0(" Done.\n");

    return TRUE;
}

/* Program internal 16x16 bit RAM in FPGA */
BOOL CPhysical::SendLUT(BYTE *data)
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    int i;
    DWORD value;

    TRACE0("Loading roms:");
    for(i=0;i<16;i++)
    {
        if (LockWait(camera,0xffff0000,((0x1c00 | data[i*2]) << 16) |
0x0040,10,0x00000040,&value)==FALSE) return FALSE;
        if (LockWait(camera,0xffff0000,((0x1d00 | data[i*2+1]) << 16) |
0x0040,10,0x00000040,&value)==FALSE) return FALSE;
        if (LockWait(camera,0xffff0000,((0x1b00 | i) << 16) |
0x0040,10,0x00000040,&value)==FALSE) return FALSE;
        TRACE1(" %d",i);
    }
    TRACE0(" Done.\n");

    return TRUE;
}

/* Send camera parameters to MCU */
BOOL CPhysical::SendParams(int *params)
{
    int camera=0; // FIX THIS FOR MULTIPLE CAMERAS
    union micromess out,in;
    BYTE coded[4];

    /* Process the first into 8 bit values */
    coded[0] = ((params[0]-100)*255)/400;
    coded[1] = ((params[1]-100)*255)/400;
    coded[2] = (params[2]*255)/100;
    coded[3] = (params[3]*255)/100;

    /* Setup the MCU packet */
    out.form.length = 9;
    out.form.type = 0x33;
```

```
    out.form.data[0] = 4;
    out.form.data[1] = 0;
    out.form.data[2] = coded[0];
    out.form.data[3] = 1;
    out.form.data[4] = coded[1];
    out.form.data[5] = 2;
    out.form.data[6] = coded[2];
    out.form.data[7] = 3;
    out.form.data[8] = coded[3];

    /* Send it */
    if (SendMicroPacket(camera,&out,&in)==FALSE) return FALSE;

    return TRUE;
}

/* Wait for a command it to be completed */
BOOL CPhysical::WaitForComplete(int camera, UINT mask, DWORD *value)
{
    int a=1,i=0;

    /* Loop for a while */
    while(a)
    {
        /* Allow other processes to run */
        Sleep(0);
        i++;
        /* Read the register */
        if (AsyncRead(camera,CAMERACOMMAND,value)==FALSE) return FALSE;
        /* Check if we met the condition */
        if ((*value & mask) == 0) a=0;
        /* Leave if we tried over 1000 */
        if (i>1000)
        {
            return FALSE;
        }
    }

    return TRUE;
}

/* Change a bit and wait for some condition on the same register */
BOOL CPhysical::LockWait(int camera, DWORD mask, DWORD ored, int pause, DWORD
maskwait, DWORD *value)
{
    /* Write the command */
    if (AsyncLock(camera,CAMERACOMMAND,mask,ored,NULL)==FALSE)
    {
        return FALSE;
    }
    else
    {
        /* Wait a bit */
        Sleep(pause);
        /* Check to see if it finished */
        if (WaitForComplete(camera, maskwait, value)==FALSE)
        {
            return FALSE;
        }
```

```
    }

    return TRUE;
}
```

D.5.12 Image Class

D.5.12.1 Image.h

```
// Image.h: interface for the CImage class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_IMAGE_H__7142DC74_2B98_11D2_A11F_0000C0059AB9__INCLUDED_)
#define AFX_IMAGE_H__7142DC74_2B98_11D2_A11F_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CImage
{
private:
    HLOCAL m_BitMem;
    HLOCAL m_BitGroups;
    int m_totalgroups;
    int m_usedgroups;
    int m_linespergroup;
    int m_width;
    int m_totallines;
    int m_usedlines;

public:
    CSize GetScrollSize();
    void CopyAll(CDC *pDC);
    BOOL CanCopyAll();
    void Paint(CDC* pDC);
    void Destroy();
    void Create(int width,int numgroups);
    int GetTotalLines();
    int GetWidth();
    BYTE *GrabLine(int line, HLOCAL *handle, BOOL expand);
    int GetValue(int x, int y);
    CImage();
    CImage(int width);
    virtual ~CImage();

};

#endif // !defined(AFX_IMAGE_H__7142DC74_2B98_11D2_A11F_0000C0059AB9__INCLUDED_)
```

D.5.12.2 Image.cpp

```
// Image.cpp: implementation of the CImage class.
//
// Create a fixed width, length variable sized gray scale image
//  with the ability to have more data added on without using
```

```
//  realloc() which is too slow.
// This class uses windows global memory to hold the image data.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Larch.h"
#include "Image.h"
#include "Imageline.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// constant definitions

#define DEFAULTWIDTH 1024 /* Default image width */
#define DEFAULTGROUPS 4 /* Default groups for initialization */

#define INCREMENTGROUPS 4 /* Number of groups to expand when more space is
required */

// Construction/Destruction
//////////////////////////////////////////////////////////////////////

/* Default Constructor, if no arguments passed, create image of default widths
and groups */
CImage::CImage()
{
    Create(DEFAULTWIDTH,DEFAULTGROUPS);
}

/* Destructor, unallocate all memory */
CImage::~CImage()
{
    Destroy();
}

/* Constructor with 1 argument (width) */
CImage::CImage(int width)
{
    Create(width,DEFAULTGROUPS);
}

/* Constructor with 2 arguments (width and number of groups */
void CImage::Create(int width, int numgroups)
{
    int i,j;
    BITMAPINFO *bmi;
//  BYTE *data;
    HLOCAL *groups;

    /* Image must be a multiple of 4 as rewuired by SetDIBitsToDevice() */
    width=max(4,width);
    m_width= (((width+3) >> 2) << 2);
    /* Compute the number of lines in 65KB blocks of RAW */
```

```
    m_linespergroup=65536/m_width;

    /* Allocate memory for bitmap header (including palette) */
    m_BitMem = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE, sizeof(BITMAPINFOHEADER)
+ (sizeof(RGBQUAD) * 256));

    /* get a pointer handle, and initialize the palette (gray scale) */
    bmi = (PBITMAPINFO) LocalLock(m_BitMem);
    for (i = 0; i < 256; i++)
    {
        bmi->bmiColors[i].rgbRed=LOBYTE(i);
        bmi->bmiColors[i].rgbGreen=LOBYTE(i);
        bmi->bmiColors[i].rgbBlue=LOBYTE(i);
        bmi->bmiColors[i].rgbReserved=0;
    }

    /* Create header */
    bmi->bmiHeader.biSize=sizeof(BITMAPINFOHEADER); // required
    bmi->bmiHeader.biWidth=m_width; // width
    bmi->bmiHeader.biHeight=m_linespergroup; // height per group
    bmi->bmiHeader.biPlanes=1; // one plane (gray scale)
    bmi->bmiHeader.biBitCount=8; // 8 bits per plane
    bmi->bmiHeader.biCompression=BI_RGB; // No compression
    bmi->bmiHeader.biSizeImage=0; // unknown at this point
//  bmi->bmiHeader.biXPelsPerMeter=GetDeviceCaps(hdc,HORZRES)/
GetDeviceCaps(hdc,HORZSIZE); // DPI not important
//  bmi->bmiHeader.biYPelsPerMeter=GetDeviceCaps(hdc,VERTRES)/
GetDeviceCaps(hdc,VERTSIZE);
    bmi->bmiHeader.biClrUsed=256; // number of palette entries used: 256
    bmi->bmiHeader.biClrImportant=256; // number of important entries: 256

    LocalUnlock(m_BitMem);

    // (SDI documents will reuse this document)

    /* setup image varibales */
    m_totalgroups = numgroups;
    m_usedgroups = 0;
    m_usedlines = -1;

    /* allocate an array to hold the pointers to each group */
    m_BitGroups = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
sizeof(groups)*m_totalgroups);
    groups = (HLOCAL*) LocalLock(m_BitGroups);

    for (j=0;j<m_totalgroups;j++)
    {
       /* allocate image data for each group */
       groups[j] = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
m_width*m_linespergroup);

// Test image for debugging (diagnoal gradient flow)
/*
       BYTE *data;

       data = (PBYTE) LocalLock(groups[j]);

       for (i=0;i<m_width*m_linespergroup;i++)
          data[i]=(i + ((i >> 8)) + j*64) & 255;
```

```
        LocalUnlock(groups[j]);
        m_usedlines=10;
*/
// End test image

    }

    LocalUnlock(m_BitGroups);
}

/* Release all memory allocated for image */
void CImage::Destroy()
{
    int j;
    HLOCAL *groups;

    /* Release all groups */
    groups = (HLOCAL*) LocalLock(m_BitGroups);
    for (j=0;j<m_totalgroups;j++)
    {
        LocalFree(groups[j]);
    }
    LocalUnlock(m_BitGroups);
    /* Release group list */
    LocalFree(m_BitGroups);

    /* Release bitmap header */
    LocalFree(m_BitMem);
}

/* Paint/Update image on screen */
void CImage::Paint(CDC * pDC)
{
    BITMAPINFO *bmi;
    BYTE *data;
    HLOCAL *groups;
    int j,gs,gf,i,k,gl,last;
    CRect reg,draw;

    /* Any lines to repaint? */
    last = m_usedlines - 1;

    if (last<0) return;

    /* Get regin to update */
    pDC->GetClipBox(&reg);

    /* Get pointers to image blocks */
    bmi = (PBITMAPINFO) LocalLock(m_BitMem);
    groups = (HLOCAL*) LocalLock(m_BitGroups);

    /* Determine which segments of images is needed to be accessed */
    gs = reg.TopLeft().y / m_linespergroup;
    gf = min(reg.BottomRight().y,last) / m_linespergroup;
    gl = last / m_linespergroup;

//  TRACE4("Refresh: %d %d %d %d\n",gs,gf,gl,reg.Width());
```

```
    /* Update the segments */
    for (j=gs;j<=gf;j++)
    {
        k = 0;
        i = m_linespergroup;

        /* Special consideration for last segment */
        if (j==gl)
        {
            i = (last % m_linespergroup) + 1;
            k = m_linespergroup - i;
        }

        data = (PBYTE) LocalLock(groups[j]);
        /* Use SetDIBitsToDevice to render bitmaps */
        SetDIBitsToDevice(pDC->m_hDC,
            reg.TopLeft().x,j*m_linespergroup,
            reg.Width(),i,
            reg.TopLeft().x,0,
            0,i,
            data+k*m_width,bmi,DIB_RGB_COLORS);
        LocalUnlock(groups[j]);
    }

    /* Release pointers */
    LocalUnlock(m_BitGroups);
    LocalUnlock(m_BitMem);

    /* Test to place test at top of window */
//  pDC->TextOut(0,0,text);
}

/* Used by UI to determine if the image can be copied */
BOOL CImage::CanCopyAll()
{
    if (m_usedlines > 0)
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

/* Copy all of the display to the clip board */
void CImage::CopyAll(CDC * pDC)
{
    TRACE0("UnComp::CopyAll()\n");

    CWaitCursor wait;

    BITMAPINFO *bmi,*gbmi;
    HGLOBAL bitmem;
    BYTE *data,*sdata;
    HANDLE test;
    HLOCAL *groups;
    int last;
```

```
    last = m_usedlines - 1;

    /* Allocate memory for uncompressed clip image */
    bitmem = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE | GMEM_DDESHARE ,
LocalSize(m_BitMem) + (m_width * m_usedlines) );

    TRACE1("UnComp::CopyAll(bitmem(size)=%d)\n",GlobalSize(bitmem));

    /* Get pointers to segments */
    bmi = (PBITMAPINFO) LocalLock(m_BitMem);
    gbmi = (PBITMAPINFO) GlobalLock(bitmem);

    /* Copy bitmap header */
    memcpy(gbmi,bmi,LocalSize(m_BitMem));

    LocalUnlock(m_BitMem);

    /* Setup more bitmap information */
    gbmi->bmiHeader.biHeight = last;
    gbmi->bmiHeader.biSizeImage = 0; //m_width * m_usedlines;
    gbmi->bmiHeader.biXPelsPerMeter=pDC->GetDeviceCaps(HORZRES) / pDC-
>GetDeviceCaps(HORZSIZE);
    gbmi->bmiHeader.biYPelsPerMeter=pDC->GetDeviceCaps(VERTRES) / pDC-
>GetDeviceCaps(VERTSIZE);

    data = (BYTE*) gbmi + LocalSize(m_BitMem);

    int j,i,k,gl;

    gl = last / m_linespergroup;

    groups = (HLOCAL*) LocalLock(m_BitGroups);

    /* Move contents of segments into clip board image */
    /*  Image is inverted for Windows, bottom to top */
    for (j=gl;j>=0;j--)
    {
        sdata = (PBYTE) LocalLock(groups[j]);

        if (j!=gl)
        {
            memcpy(data,sdata,LocalSize(groups[j]));
            data += LocalSize(groups[j]);
        }
        else
        {
            i = (last % m_linespergroup) + 1;
            k = m_linespergroup - i;

            memcpy(data,sdata+(k*m_width),i*m_width);
            data += (i*m_width);

        }

        LocalUnlock(groups[j]);

    }

    /* Release pointers */
```

```
    LocalUnlock(m_BitGroups);

    GlobalUnlock(bitmem);

    /* Set clipboard to a device independant bitmap */
    /* Don't have to render it for other applications */
    test = SetClipboardData(CF_DIB, bitmem);

    return;
}

/* Returns total lines in image, to reviel protected data */
int CImage::GetTotalLines()
{
    return m_totallines;
}

/* Returns width of image, to reviel protected data */
int CImage::GetWidth()
{
    return m_width;
}

/* Grab a pointer to a single line from group of segments */
BYTE *CImage::GrabLine(int line, HLOCAL *handle, BOOL expand)
{
    int i,j;
    HLOCAL *groups;
    BYTE *pointer;
    DWORD err;

    *handle = NULL;

    /* May have to grow segemtns if the line doesn't exists */
    if (line >= m_totalgroups*m_linespergroup)
    {
        /* If expand is set, grow segments if necessary */
        if (expand==TRUE)
        {
            i = line/m_linespergroup + INCREMENTGROUPS;

            /* Get pointers */
            m_BitGroups = LocalReAlloc(m_BitGroups,  sizeof(groups)*i,
LMEM_ZEROINIT | LMEM_MOVEABLE);
            ASSERT(m_BitGroups);

            groups = (HLOCAL*) LocalLock(m_BitGroups);
            ASSERT(groups);

            /* Create new segments */
            for (j=m_totalgroups;j<i;j++)
            {
                groups[j] = LocalAlloc(LMEM_ZEROINIT | LMEM_MOVEABLE,
m_width*m_linespergroup);
                err = GetLastError();
                ASSERT(groups[j]);
            }

            /* Release potiners */
```

```
            LocalUnlock(m_BitGroups);

            m_totalgroups=i;
        }
        else
        {
            return NULL;
        }
    }

    /* Return NULL line doesn't exist */
    if (line > m_usedlines)
    {
        if (expand == FALSE)
        {
            return NULL;
        }
    }

    /* Get pointers */
    groups = (HLOCAL*) LocalLock(m_BitGroups);

    /* Get pointer to inside of segment */
    *handle = groups[line/m_linespergroup];
    pointer = (PBYTE) LocalLock(*handle)+(m_linespergroup - 1 - (line %
m_linespergroup))*m_width;

    /* Release one of the pointers, the reset will be release later */
    LocalUnlock(m_BitGroups);

    /* Update object data */
    m_totallines = max(line,m_totallines);
    m_usedlines = max(line,m_usedlines);

    /* If this pointer is bad, something is serious wrong, stop the application */
    if (pointer==NULL)
    {
        AfxAbort();
    }

    return pointer;
}

/* Get a gray value from the segments */
int CImage::GetValue(int x, int y)
{
    CImageline *line;
    int value;

    /* Grab a pointer to a line */
    line = new CImageline(this, y, FALSE);

    /* The we are off of it, return -1 */
    if (line->m_pointer==NULL)
    {
        value=-1;
    }
    else
    {
```

```
        value=line->m_pointer[x];
    }

    delete line;

    return value;
}

/* Create a CSize element with image size and height */
CSize CImage::GetScrollSize()
{
    CSize size(GetWidth(),max(m_usedlines,0));

    return size;
}
```

D.5.13 Imageline Class

D.5.13.1 Imageline.h

```
// Imageline.h: interface for the CImageline class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_IMAGELINE_H__7142DC75_2B98_11D2_A11F_0000C0059AB9__INCLUDED_)
#define AFX_IMAGELINE_H__7142DC75_2B98_11D2_A11F_0000C0059AB9__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CImageline
{
public:
    BYTE *m_pointer;
private:
    HLOCAL m_handle;

public:
    CImageline(CImage *base, int line, BOOL expand);
    virtual ~CImageline();

};

#endif //
!defined(AFX_IMAGELINE_H__7142DC75_2B98_11D2_A11F_0000C0059AB9__INCLUDED_)
```

D.5.13.2 Imageline.cpp

```
// Imageline.cpp: implementation of the CImageline class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Larch.h"
#include "Image.h"
#include "Imageline.h"
```

```
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CImageline::CImageline(CImage *base, int line, BOOL expand)
{
    /* Grab a line using the CImage class */
    m_pointer=base->GrabLine(line, &m_handle, expand);
}

CImageline::~CImageline()
{
    /* Release the global memory on destruction */
    if (m_pointer!=NULL)
    {
        LocalUnlock(m_handle);
    }
    m_pointer=NULL;
}
```

D.5.14 CCamParams Class

D.5.14.1 CamParams.h

```
#if !defined(AFX_CAMPARAMS_H__616C9683_464A_11D2_A13C_0000C0059AB9__INCLUDED_)
#define AFX_CAMPARAMS_H__616C9683_464A_11D2_A13C_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CamParams.h : header file
//

//////////////////////////////////////////////////////////////////////////////
// CCamParams dialog

class CCamParams : public CDialog
{
// Construction
public:
    CCamParams(CWnd* pParent = NULL);   // standard constructor
    void SetParams(int *params);
    void GetParams(int *params);

private:
    CStatic *m_textgain1,*m_textgain2,*m_textoffset1,*m_textoffset2;
    CEdit *m_editgain1,*m_editgain2,*m_editoffset1,*m_editoffset2;
    CSpinButtonCtrl *m_spingain1,*m_spingain2,*m_spinoffset1,*m_spinoffset2;
    int m_params[4];
```

```
// Dialog Data
    //{{AFX_DATA(CCamParams)
    enum { IDD = IDD_CAMERA };
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CCamParams)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CCamParams)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_CAMPARAMS_H__616C9683_464A_11D2_A13C_0000C0059AB9__INCLUDED_)
```

D.5.14.2 CamParams.cpp

```
// CamParams.cpp : implementation file
//
// Dialog control for camera internally MCU settings.

#include "stdafx.h"
#include "Larch.h"
#include "CamParams.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCamParams dialog


CCamParams::CCamParams(CWnd* pParent /*=NULL*/)
    : CDialog(CCamParams::IDD, pParent)
{
    //{{AFX_DATA_INIT(CCamParams)
    //}}AFX_DATA_INIT
}
```

```
void CCamParams::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCamParams)
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCamParams, CDialog)
    //{{AFX_MSG_MAP(CCamParams)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCamParams message handlers

BOOL CCamParams::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    /* Get pointers to the controls */
    m_textgain1 = (CStatic*)GetDlgItem(IDC_STATICGAIN1);
    m_editgain1 = (CEdit*)GetDlgItem(IDC_EDITGAIN1);
    m_spingain1 = (CSpinButtonCtrl*)GetDlgItem(IDC_SPINGAIN1);
    m_textgain2 = (CStatic*)GetDlgItem(IDC_STATICGAIN2);
    m_editgain2 = (CEdit*)GetDlgItem(IDC_EDITGAIN2);
    m_spingain2 = (CSpinButtonCtrl*)GetDlgItem(IDC_SPINGAIN2);
    m_textoffset1 = (CStatic*)GetDlgItem(IDC_STATICOFFSET1);
    m_editoffset1 = (CEdit*)GetDlgItem(IDC_EDITOFFSET1);
    m_spinoffset1 = (CSpinButtonCtrl*)GetDlgItem(IDC_SPINOFFSET1);
    m_textoffset2 = (CStatic*)GetDlgItem(IDC_STATICOFFSET2);
    m_editoffset2 = (CEdit*)GetDlgItem(IDC_EDITOFFSET2);
    m_spinoffset2 = (CSpinButtonCtrl*)GetDlgItem(IDC_SPINOFFSET2);

    /* Set the text limit on the fields */
    m_editgain1->LimitText(3);
    m_editgain2->LimitText(3);
    m_editoffset1->LimitText(3);
    m_editoffset2->LimitText(3);

    /* Set ranges on the spin controls */
    m_spingain1->SetRange(100,500);
    m_spingain2->SetRange(100,500);
    m_spinoffset1->SetRange(0,100);
    m_spinoffset2->SetRange(0,100);

    char temp[10];

    /* Set the text fields */
    m_editgain1->SetWindowText(itoa(m_params[0],temp,10));
    m_editgain2->SetWindowText(itoa(m_params[1],temp,10));
    m_editoffset1->SetWindowText(itoa(m_params[2],temp,10));
    m_editoffset2->SetWindowText(itoa(m_params[3],temp,10));

    /* Set spin control values */
    m_spingain1->SetPos(m_params[0]);
    m_spingain2->SetPos(m_params[1]);
```

```
    m_spinoffset1->SetPos(m_params[2]);
    m_spinoffset2->SetPos(m_params[3]);

    /* Disable the Second channel functions since we don't have it */
    m_textgain2->EnableWindow(FALSE);
    m_editgain2->EnableWindow(FALSE);
    m_spingain2->EnableWindow(FALSE);
    m_textoffset2->EnableWindow(FALSE);
    m_editoffset2->EnableWindow(FALSE);
    m_spinoffset2->EnableWindow(FALSE);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CCamParams::OnOK()
{
    // TODO: Add extra validation here
    int value,check=0;
    char temp[10];
    int params[4];

    /* Check inputs to make sure they are in range */
    m_editgain1->GetWindowText(temp,9);
    value = atoi(temp);
    if (value < 100 || value > 500)
    {
        MessageBox("Gain Multiplier Channel 1 must be greater than 99 and less
than 501.","Parameter Error",MB_ICONEXCLAMATION);
        check++;
    }
    else params[0] = value;
/*
    m_editgain2->GetWindowText(temp,9);
    value = atoi(temp);
    if (value < 100 || value > 500)
    {
        MessageBox("Gain Multiplier Channel 2 must be greater than 99 and less
than 501.","Parameter Error",MB_ICONEXCLAMATION);
        check++;
    }
    else params[1] = value;
*/
    m_editoffset1->GetWindowText(temp,9);
    value = atoi(temp);
    if (value < 0 || value > 100)
    {
        MessageBox("Offset Channel 1 must be greater than equal to 0 and less than
101.","Parameter Error",MB_ICONEXCLAMATION);
        check++;
    }
    else params[2] = value;
/*
    m_editoffset2->GetWindowText(temp,9);
    value = atoi(temp);
    if (value < 0 || value > 100)
    {
        MessageBox("Offset Channel 1 must be greater than equal to 0 and less than
101.","Parameter Error",MB_ICONEXCLAMATION);
```

```
        check++;
    }
    else params[3] = value;
*/
    /* If ok, return from dialog */
    if (!check)
    {
        int i;
        for (i=0;i<4;i++) m_params[i]=params[i];
        CDialog::OnOK();
    }
}

/* Return paramters to calling function */
void CCamParams::GetParams(int *params)
{
    int i;
    for (i=0;i<4;i++)
    {
        params[i]=m_params[i];
    }
}

/* Set parameters from calling function */
void CCamParams::SetParams(int *params)
{
    int i;
    for (i=0;i<4;i++)
    {
        m_params[i]=params[i];
    }
}
```

D.5.15 CParams Class

D.5.15.1 Params.h

```
#if !defined(AFX_PARAMS_H__A1FA25B8_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
#define AFX_PARAMS_H__A1FA25B8_40C9_11D2_A13A_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Params.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CParams dialog

#define GETPARAMS_MAX 8

class CParams : public CDialog
{
// Construction
private:
    CStatic *m_text[GETPARAMS_MAX];
    CEdit *m_edit[GETPARAMS_MAX];
    CSpinButtonCtrl *m_spin[GETPARAMS_MAX];
```

```
    BYTE *m_prcblock,*m_rules;
    BYTE m_total;
    int m_params[GETPARAMS_MAX];

public:
    void SetParams(int *params);
    void GetParams(int *params);
    CParams(CWnd* pParent = NULL, BYTE *prcblock = NULL);   // standard
constructor

// Dialog Data
    //{{AFX_DATA(CParams)
    enum { IDD = IDD_PARAMS };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CParams)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CParams)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_PARAMS_H__A1FA25B8_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
```

D.5.15.2 Params.cpp

```
// Params.cpp : implementation file
//
// Read paramters from video processor data and set up a
//  dialog box with rules to follow.

#include "stdafx.h"
#include "Larch.h"
#include "Params.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CParams dialog


CParams::CParams(CWnd* pParent /*=NULL*/, BYTE *prcblock)
    : CDialog(CParams::IDD, pParent)
{
    //{{AFX_DATA_INIT(CParams)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    m_prcblock = prcblock;
}


void CParams::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CParams)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CParams, CDialog)
    //{{AFX_MSG_MAP(CParams)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CParams message handlers

BOOL CParams::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here
    /* Create array of resource IDs for dialog controls */
    static int i_text[GETPARAMS_MAX] = {IDC_TEXT1, IDC_TEXT2, IDC_TEXT3,
IDC_TEXT4,
        IDC_TEXT5, IDC_TEXT6, IDC_TEXT7, IDC_TEXT8};
    static int i_edit[GETPARAMS_MAX] = {IDC_EDIT1, IDC_EDIT2, IDC_EDIT3,
IDC_EDIT4,
        IDC_EDIT5, IDC_EDIT6, IDC_EDIT7, IDC_EDIT8};
    static int i_spin[GETPARAMS_MAX] = {IDC_SPIN1, IDC_SPIN2, IDC_SPIN3,
IDC_SPIN4,
        IDC_SPIN5, IDC_SPIN6, IDC_SPIN7, IDC_SPIN8};

    int i,j,k;

    BYTE *point = m_prcblock;

    /* Set dialog title based on video processor name */
    SetDlgItemText(IDC_TITLE,(LPCTSTR)point);
    point += (strlen((LPCTSTR)point)+1);

    m_total = *point;
    point++;

    // point is at default values
```

```
// point += m_total*sizeof(int);

    for (i=0;i<GETPARAMS_MAX;i++)
    {
        /* Get pointers to the resources */
        m_text[i] = (CStatic*)GetDlgItem(i_text[i]);
        m_edit[i] = (CEdit*)GetDlgItem(i_edit[i]);
        m_spin[i] = (CSpinButtonCtrl*)GetDlgItem(i_spin[i]);

        /* Set the text limit to 5 chars and zero out the fields */
        m_edit[i]->LimitText(5);
        m_edit[i]->SetWindowText("");

        /* Set the maximum range from 0 to 65535 on the spin controls */
        m_spin[i]->SetRange(0,65535);
        /* Set the actual parameters */
        m_spin[i]->SetPos(m_params[i]);
    }

    for (i=0;i<m_total;i++)
    {
        /* Enable the controls provided by the video processor */
        m_text[i]->EnableWindow(TRUE);
        m_edit[i]->EnableWindow(TRUE);
        m_spin[i]->EnableWindow(TRUE);

        /* Set the text of the controls */
        m_text[i]->SetWindowText((LPCTSTR)point);
        point += (strlen((LPCTSTR)point)+1);
    }

    /* Set the rest of the controls to 0 */
    for (i=m_total;i<GETPARAMS_MAX;i++)
    {
        m_spin[i]->SetPos(0);
    }

    int up,low;
    DWORD value;
    m_rules = point;

    /* Set maximum and minimums of the controls based on the */
    /*  parameter rules */
    while (*point!=255)
    {
        i = *point;
        point++;
        value = m_spin[i]->GetRange();
        low = HIWORD(value);
        up = LOWORD(value);

        j = *point;
        point++;

        k = *point + (*(point+1) << 8);
        point += 2;

        switch (j)
```

```
        {
            case 0: // greater than static
                low = k+1;
                break;
            case 1: // lesser than static
                up = k-1;
                break;
            case 2: // greater than equal static
                low = k;
                break;
            case 3: // lesser than equal static
                up = k;
                break;
        }

        m_spin[i]->SetRange(low,up);
        m_spin[i]->SetPos(m_params[i]);
    }

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

void CParams::OnOK()
{
    // TODO: Add extra validation here

    /* User pressed OK */
    int i,j,k;
    BYTE *point = m_rules;
    int value1,value2;
    int error;
    char temp[10];
    CString title1,title2,problem,message;

    /* Check the rules based on the users input */
    while (*point!=255)
    {
        error = 0;
        i = *point;
        point++;
        value1 = m_spin[i]->GetPos();
        if (HIWORD(value1))
        {
            m_edit[i]->GetWindowText(temp,9);
            value1 = atoi(temp);
        }
        else
        {
            value1 = LOWORD(value1);
        }
        j = *point;
        point++;

        k = *point + (*(point+1) << 8);
        point += 2;

        switch (j)
        {
```

```
case 0: // greater than static
    if (value1 <= k)
    {
        problem = "greater than";
        error = 2;
    }
    break;
case 1: // lesser than static
    if (value1 >= k)
    {
        problem = "lesser than";
        error = 2;
    }
    break;
case 2: // greater than equal static
    if (value1 < k)
    {
        problem = "greater than equal to";
        error = 2;
    }
    break;
case 3: // lesser than equal static
    if (value1 > k)
    {
        problem = "lesser than equal to";
        error = 2;
    }
    break;
case 4: // greater than index
    value2 = m_spin[k]->GetPos();
    if (value1 <= value2)
    {
        problem = "greater than";
        error = 1;
    }
    break;
case 5: // lesser than index
    value2 = m_spin[k]->GetPos();
    if (value1 >= value2)
    {
        problem = "lesser than";
        error = 1;
    }
    break;
case 6: // greater than equal index
    value2 = m_spin[k]->GetPos();
    if (value1 < value2)
    {
        problem = "greater than or equal to";
        error = 1;
    }
    break;
case 7: // lesser than equal index
    value2 = m_spin[k]->GetPos();
    if (value1 > value2)
    {
        problem = "lesser than or equal to";
        error = 1;
    }
```

```
                break;
            }
            /* Display a message if there is a problem */
            if (error)
            {
                m_text[i]->GetWindowText(title1);
                if (error == 1)
                {
                    m_text[k]->GetWindowText(title2);
                    title2 = "`" + title2 + "'";
                }
                if (error == 2)
                {
                    itoa(k,temp,10);
                    title2 = temp;
                }
                message = "`" + title1 + "' must be " + problem + " " + title2 + ".";
                MessageBox(message,"Parameter Error",MB_ICONEXCLAMATION);
                break;
            }

        }

        /* No problem, set the return parameters and leave */
        if (error==0)
        {
            for (i=0;i<GETPARAMS_MAX;i++)
            {
                m_params[i] = m_spin[i]->GetPos();
            }

            CDialog::OnOK();
        }

}

/* Get parameters for calling function */
void CParams::GetParams(int *params)
{
    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        params[i]=m_params[i];
    }
}

/* Save parameters from calling function */
void CParams::SetParams(int *params)
{
    int i;
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        m_params[i]=params[i];
    }
}
```

D.5.16 External Processor Interface

D.5.16.1 Processor.h

```
// The following ifdef block is the standard way of creating macros which make
exporting
// from a DLL simpler. All files within this DLL are compiled with the
PROCESSOR_EXPORTS
// symbol defined on the command line. this symbol should not be defined on any
project
// that uses this DLL. This way any other project whose source files include this
file see
// PROCESSOR_API functions as being imported from a DLL, wheras this DLL sees
symbols
// defined with this macro as being exported.
#ifdef PROCESSOR_EXPORTS
#define PROCESSOR_API __declspec(dllexport)
#else
#define PROCESSOR_API __declspec(dllimport)
#endif

/* Maximum number of parameters */
#define GETPARAMS_MAX 8

/* Used by AddData to move information about decompressing the current line */
typedef struct _lineinfo
{
    int line;
    BYTE *data;
} lineinfo;

/* Return the amount of private data the video processing data will need */
/*  This is to avoid having the DLL require it's on heap */
PROCESSOR_API DWORD PrivateDataSize();
/* Called to initialize video processor with private data pointer */
PROCESSOR_API int InitData(void *p);
/* Called to initialize video processor before capture */
PROCESSOR_API void StartData(void *p);
/* Called to decompress incoming data */
PROCESSOR_API BOOL AddData(void *p, BYTE **indata, int *total, void *line);
/* Returns pointer to FPGA bitmap and size */
PROCESSOR_API BYTE* GetFPGABitmap(int *size);
/* Returns video processor name */
PROCESSOR_API DWORD GetTitle(char *name,DWORD size);
/* Returns default video processor parameters */
PROCESSOR_API void GetDefaultParams(void *p, int *params);
/* Generates coded 8-bit values from parameters */
PROCESSOR_API DWORD Generate(void *p, int *params, BYTE* coded);
/* Returns parameter block for rules on processing parameters */
PROCESSOR_API void GetParamBlock(void *p, BYTE *block);
```

D.5.16.2 ProcessorCode.h

```
/* This code is global and included into all the video processor modules */
#include "string.h"

#include "Processor.h"
```

```
/* Build the parameter block */
static struct _paramblk
{
    char name[sizeof(PBNAME)];
    unsigned char num[1];
    char option1[sizeof(PBOPT1)];
#ifdef PBOPT2
    char option2[sizeof(PBOPT2)];
#endif
#ifdef PBOPT3
    char option3[sizeof(PBOPT3)];
#endif
#ifdef PBOPT4
    char option4[sizeof(PBOPT4)];
#endif
#ifdef PBOPT5
    char option5[sizeof(PBOPT5)];
#endif
#ifdef PBOPT6
    char option6[sizeof(PBOPT6)];
#endif
#ifdef PBOPT7
    char option7[sizeof(PBOPT7)];
#endif
#ifdef PBOPT8
    char option8[sizeof(PBOPT8)];
#endif
    struct _rules
    {
        unsigned char index;
        unsigned char rule;
        unsigned char value1;
        unsigned char valueh;
    } rules[PBNUMRULES];
    unsigned char end;
} pbdata = {
    {PBNAME},
    {PBNUMOPT},
    {PBOPT1},
#ifdef PBOPT2
    {PBOPT2},
#endif
#ifdef PBOPT3
    {PBOPT3},
#endif
#ifdef PBOPT4
    {PBOPT4},
#endif
#ifdef PBOPT5
    {PBOPT5},
#endif
#ifdef PBOPT6
    {PBOPT6},
#endif
#ifdef PBOPT7
    {PBOPT7},
#endif
#ifdef PBOPT8
    {PBOPT8},
```

```
#endif
    {PBRULES},
    255
};

/* Handle for the DLL */
HANDLE Module;

/* Windows stuff for initializing the DLL */
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
            Module = hModule;
            break·

        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}

/* Return the amount of private data that will be needed */
PROCESSOR_API DWORD PrivateDataSize()
{
    /* Return the size of the structure */
    return sizeof(PD);
}

/* Get the FPGA bitmap from the Windows resource */
PROCESSOR_API BYTE* GetFPGABitmap(int *size)
{
    HGLOBAL global;
    HRSRC rsrc;
    LPVOID lp;
    BYTE *returned;

    *size = 0;

    /* Get a handle to the resource */
    rsrc = FindResource(Module,(LPCTSTR)MAKELONG(IDR_FPGA,0),"FPGA");
    if (!rsrc) return NULL;

    /* Get a handle to the object */
    global = LoadResource(Module,rsrc);
    if (!global) return NULL;

    /* Get a pointer and size */
    lp = LockResource(global);
    *size = SizeofResource(Module,rsrc);
    if (!size) return NULL;

    /* Can seem to unload the memory, I think this is a Windows */
```

```
    /*  memory leak problem */
    returned = lp;

    return returned;
}

/* Return the title of the video processor */
PROCESSOR_API DWORD GetTitle(char *name,DWORD size)
{
    DWORD a;

    a = min(strlen(PBNAME),size);

    strncpy(name,PBNAME,a);
    name[a]=0;

    return a;
}

/* Return the default parameters */
PROCESSOR_API void GetDefaultParams(void *p, int *params)
{
    PD *pd = (PD*)p;
    int i;
    int paramdefs[PBNUMOPT]={PBOPTDEFS};

    for (i=0;i<GETPARAMS_MAX;i++)
    {
        if (i<PBNUMOPT) params[i]=paramdefs[i];
        else params[i]=0;
    }

    return;
}

/* Copy parameters block into specified memory */
PROCESSOR_API void GetParamBlock(void *p, BYTE *block)
{
    memcpy(block,&pbdata,sizeof(pbdata));

    return;
}
```

D.6 Focus Processor Code

D.6.1 Project file

D.6.1.1 Focus.dsp

```
# Microsoft Developer Studio Project File - Name="Focus" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=Focus - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Focus.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Focus.mak" CFG="Focus - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Focus - Win32 Release" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE "Focus - Win32 Debug" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Focus - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "FOCUS_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS"
/D "_USRDLL" /D "FOCUS_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
```

```
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386

!ELSEIF  "$(CFG)" == "Focus - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "FOCUS_EXPORTS" /Yu"stdafx.h" /FD /GZ /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
D "_MBCS" /D "_USRDLL" /D "PROCESSOR_EXPORTS" /FD /GZ /c
# SUBTRACT CPP /YX /Yc /Yu
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /out:"Debug/Focus.vpr" /pdbtype:sept

!ENDIF

# Begin Target

# Name "Focus - Win32 Release"
# Name "Focus - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\Focus.c
# End Source File
# Begin Source File

SOURCE=.\StdAfx.c
```

```
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\resource.h
# End Source File
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\Focus.rc
# End Source File
# Begin Source File

SOURCE=.\fpga.bin
# End Source File
# End Group
# End Target
# End Project
```

D.6.2 MFC Files

D.6.2.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
#define AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


// Insert your headers here
#define WIN32_LEAN_AND_MEAN// Exclude rarely-used stuff from Windows headers

#include <windows.h>
// #define _AFX_NOFORCE_LIBS
// #include "afxwin.h"

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.
```

```
#endif //
!defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
```

D.6.2.2 StdAfx.c

```
// stdafx.cpp : source file that includes just the standard includes
//  Lineup.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

D.6.3 Resource Files

D.6.3.1 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Lineup.rc
//
#define IDR_FPGA                        101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.6.3.2 Focus.rc

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

/////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

/////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
```

```
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

/////////////////////////////////////////////////////////////////////////////
//
// FPGA
//

IDR_FPGA                FPGA    DISCARDABLE     "fpga.bin"

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

D.6.4 Focus Code Files

D.6.4.1 Focus.c

```
// Focus.c : Focus processor for second generation camera
//
```

```
#include "stdafx.h"

#include "resource.h"

/* Define video processor name */
#define PBNAME "Focus Processor"
/* Number of parameters */
#define PBNUMOPT 2
/* Their defaults */
#define PBOPTDEFS 512,1024
/* Their titles */
#define PBOPT1 "Start Pixel"
#define PBOPT2 "Length of Line"
/* The number of rules */
#define PBNUMRULES 4
/* The rules themselves */
#define PBRULES {0,2,1,0}, {0,3,0,8}, {1,2,1,0}, {1,3,0,8}

/* Set private data */
typedef struct _PD
{
    BOOL synced;
    int currentline;
    int currentoffset;
    int linelength;
} PD;

/* Include global functions */
#include "..\Larch\ProcessorCode.h"

/* Generate 8 bytes coded paramaters for FPGA */
PROCESSOR_API DWORD Generate(void *p, int *params, BYTE* coded)
{
    PD *pd = (PD*)p;
    int i,a;

    /* Zero out all values initially */
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        coded[i]=0;
    }

    /* Start pixel */
    a = params[0] - 1;
    coded[0] = a & 255;
    coded[2] = a >> 8;

    /* Line length */
    a = (params[0] - 1) + (params[1] - 1);
    coded[1] = a & 255;
    coded[2] = coded[2] | ((a >> 8) << 3);

    pd->linelength = params[1];

    return pd->linelength;
}

/* Initialize processor data */
```

```
PROCESSOR_API int InitData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;
    pd->linelength = 1024;

    return sizeof(pbdata);
}

/* Initalize data before capture */
PROCESSOR_API void StartData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;

    return;
}

/* Decompress/Decode video data from camera */
PROCESSOR_API BOOL AddData(void *p, BYTE **indata, int *total, void *l)
{
    PD *pd = (PD*)p;
    lineinfo *li = (lineinfo*)l;
    int i=0;
    BYTE *data = *indata;

    /* Look for sync bit */
    if (pd->synced == FALSE)
    {
        while (i<*total)
        {
            if ((data[i] & 4))
            {
                pd->currentoffset = 0;
                pd->synced = TRUE;
                break;
            }
            i+=4;
        }
    }

    /* Get current line */
    pd->currentline = li->line;

    /* Loop through all the new data */
    while(i<*total)
    {
        /* Go through a complete line */
        while ((pd->currentoffset != pd->linelength) && (i<*total))
        {
            /* Skip flag byte (only 3 bits) */
            if ((i & 3) != 0)
            {
```

```
            li->data[pd->currentoffset]=data[i];
            pd->currentoffset++;
        }
        i++;
    }
    /* End of line */
    if (pd->currentoffset == pd->linelength)
    {
        /* Finished it properly */
        pd->synced = FALSE;
        pd->currentoffset = 0;
        pd->currentline = pd->currentline + 1;

        /* Find the sync code */
        while (i<*total)
        {
            if ((i & 3) == 0)
            {
                if ((data[i] & 4))
                {
                    pd->synced = TRUE;
                    break;
                }
            }
            i++;
        }

        /* Update data processed */
        *indata += i;
        *total -= i;

        /* Update the line pointer */
        li->line=pd->currentline;
        return TRUE;
    }
    else
    {
        /* Line not done */
        li->line=pd->currentline;
        return FALSE;
    }
}

/* Data done, line not likely done */
return FALSE;
}
```

D.7 Lineup Processor Code

D.7.1 Project file

D.7.1.1 Lineup.dsp

```
# Microsoft Developer Studio Project File - Name="Lineup" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=Lineup - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Lineup.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Lineup.mak" CFG="Lineup - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Lineup - Win32 Release" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE "Lineup - Win32 Debug" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Lineup - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "LINEUP_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS"
/D "_USRDLL" /D "LINEUP_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
```

```
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386

!ELSEIF  "$(CFG)" == "Lineup - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "LINEUP_EXPORTS" /Yu"stdafx.h" /FD /GZ /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /I "D:\Adaptec\dev\include" /D "WIN32"
/D "_DEBUG" /D "_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "PROCESSOR_EXPORTS" /FD /GZ
/c
# SUBTRACT CPP /YX /Yc /Yu
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /out:"Debug/Lineup.vpr" /pdbtype:sept

!ENDIF

# Begin Target

# Name "Lineup - Win32 Release"
# Name "Lineup - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\Lineup.c
# End Source File
# Begin Source File
```

```
SOURCE=.\StdAfx.c
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\resource.h
# End Source File
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\Lineup.rc
# End Source File
# End Group
# End Target
# End Project
```

D.7.2 MFC Files

D.7.2.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
#define AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


// Insert your headers here
#define WIN32_LEAN_AND_MEAN// Exclude rarely-used stuff from Windows headers

#include <windows.h>
// #define _AFX_NOFORCE_LIBS
// #include "afxwin.h"

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
```

D.7.2.2 StdAfx.c

```
// stdafx.cpp : source file that includes just the standard includes
//  Lineup.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

D.7.3 Resource Files

D.7.3.1 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Lineup.rc
//
#define IDR_FPGA                        101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.7.3.2 Lineup.rc

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32
```

```
//////////////////////////////////////////////////////////////////////////////
//
// FPGA
//

IDR_FPGA                FPGA    DISCARDABLE     "fpga.bin"

#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

#endif    // English (U.S.) resources
//////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


//////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

D.7.4 Lineup Code Files

D.7.4.1 Lineup.c

```
// Lineup.c : Lineup processor for second generation camera
//

#include "stdafx.h"
```

```
#include "resource.h"

/* Define video processor name */
#define PBNAME "Lineup Processor"
/* Number of parameters */
#define PBNUMOPT 2
/* Their defaults */
#define PBOPTDEFS 128,128
/* Their titles */
#define PBOPT1 "Pixels on the Left"
#define PBOPT2 "Pixels on the Right"
/* The number of rules */
#define PBNUMRULES 4
/* The rules themselves */
#define PBRULES {0,2,0,0}, {0,3,0,2}, {1,2,0,0}, {1,3,0,2}

/* Set private data */
typedef struct _PD
{
    BOOL synced;
    int currentline;
    int currentoffset;
    int linelength;
} PD;

/* Include global functions */
#include "..\Larch\ProcessorCode.h"

/* Generate 8 bytes coded paramaters for FPGA */
PROCESSOR_API DWORD Generate(void *p, int *params, BYTE* coded)
{
    PD *pd = (PD*)p;
    int i,a;

    /* Zero out all values initially */
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        coded[i]=0;
    }

    /* Pixels on left */
    a = params[0];
    coded[0] = a & 255;
    coded[2] = a >> 8;

    /* Pixels on right */
    a = 2047-params[1];
    coded[1] = a & 255;
    coded[2] = coded[2] | ((a >> 8) << 3);

    pd->linelength = min(params[0]+params[1],2048);

    return pd->linelength;
}

/* Initialize processor data */
PROCESSOR_API int InitData(void *p)
{
    PD *pd = (PD*)p;
```

```
    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;
    pd->linelength = 1024;

    return sizeof(pbdata);
}

/* Initalize data before capture */
PROCESSOR_API void StartData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;

    return;
}

/* Decompress/Decode video data from camera */
PROCESSOR_API BOOL AddData(void *p, BYTE **indata, int *total, void *l)
{
    PD *pd = (PD*)p;
    lineinfo *li = (lineinfo*)l;
    int i=0;
    BYTE *data = *indata;

    /* Look for sync bit */
    if (pd->synced == FALSE)
    {
        while (i<*total)
        {
            if ((data[i+3] & 4))
            {
                pd->currentoffset = 0;
                pd->synced = TRUE;
                break;
            }
            i+=4;
        }
    }

    /* Get current line */
    pd->currentline = li->line;

    /* Loop through all the new data */
    while(i<*total)
    {
        /* Go through a complete line */
        while ((pd->currentoffset != pd->linelength) && (i<*total))
        {
            /* Skip flag byte (only 3 bits) */
            if ((i & 3) != 0)
            {
                li->data[pd->currentoffset]=data[i];
                pd->currentoffset++;
            }
```

```
            i++;
        }
        /* End of line */
        if (pd->currentoffset == pd->linelength)
        {
            /* Finished it properly */
            pd->synced = FALSE;
            pd->currentoffset = 0;
            pd->currentline = pd->currentline + 1;

            /* Find the sync code */
            while (i<*total)
            {
                if ((i & 3) == 0)
                {
                    if ((data[i] & 4))
                    {
                        pd->synced = TRUE;
                        break;
                    }
                }
                i++;
            }

            /* Update data processed */
            *indata += i;
            *total -= i;

            /* Update the line pointer */
            li->line=pd->currentline;
            return TRUE;
        }
        else
        {
            /* Line not done */
            li->line=pd->currentline;
            return FALSE;
        }
    }

    /* Data done, line not likely done */
    return FALSE;
}
```

D.8 Minimum/Maximum Processor Code

D.8.1 Project file

D.8.1.1 Minmax.dsp

```
# Microsoft Developer Studio Project File - Name="Minmax" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=Minmax - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Minmax.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Minmax.mak" CFG="Minmax - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Minmax - Win32 Release" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE "Minmax - Win32 Debug" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Minmax - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "MINMAX_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS"
/D "_USRDLL" /D "MINMAX_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
```

```
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386

!ELSEIF  "$(CFG)" == "Minmax - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "MINMAX_EXPORTS" /Yu"stdafx.h" /FD /GZ  /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
D "_MBCS" /D "_USRDLL" /D "PROCESSOR_EXPORTS" /FD /GZ /c
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /out:"Debug/Minmax.vpr" /pdbtype:sept

!ENDIF

# Begin Target

# Name "Minmax - Win32 Release"
# Name "Minmax - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\Minmax.c
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
```

```
# Begin Source File

SOURCE=.\resource.h
# End Source File
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\Minmax.rc
# End Source File
# End Group
# End Target
# End Project
```

D.8.2 MFC Files

D.8.2.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
#define AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


// Insert your headers here
#define WIN32_LEAN_AND_MEAN// Exclude rarely-used stuff from Windows headers

#include <windows.h>
// #define _AFX_NOFORCE_LIBS
// #include "afxwin.h"

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.

#endif //
!defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
```

D.8.2.2 StdAfx.c

```
// stdafx.cpp : source file that includes just the standard includes
//  Lineup.pch will be the pre-compiled header
```

```
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

D.8.3 Resource Files

D.8.3.1 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Lineup.rc
//
#define IDR_FPGA                        101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.8.3.2 Minmax.rc

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

////////////////////////////////////////////////////////////////////////////
//
// FPGA
//
```

```
IDR_FPGA                FPGA    DISCARDABLE     "fpga.bin"

#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

#endif    // English (U.S.) resources
//////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


//////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

D.8.4 Minmax Code Files

D.8.4.1 Minmax.c

```
// Minmax.c : Minimum/Maximum processor for second generation camera
//

#include "stdafx.h"

#include "resource.h"

/* Define video processor name */
#define PBNAME "Minimum/Maximum Processor"
/* Number of parameters */
```

```
#define PBNUMOPT 5
/* Their defaults */
#define PBOPTDEFS 1,2048,5,250,128
/* Their titles */
#define PBOPT1 "Start Pixel"
#define PBOPT2 "Length of Line"
#define PBOPT3 "Minimum Pixel Value"
#define PBOPT4 "Maximum Pixel Value"
#define PBOPT5 "Synthesised Background"
/* The number of rules */
#define PBNUMRULES 11
/* The rules themselves */
#define PBRULES {0,2,1,0}, {0,3,0,8}, {1,2,1,0}, {1,3,0,8}, {2,2,0,0},
{2,3,255,0}, {3,2,0,0}, {3,3,255,0}, {2,5,3,0}, {4,2,0,0}, {4,3,255,0}

/* Set private data */
typedef struct _PD
{
    BOOL synced;
    int currentline;
    int currentoffset;
    int linelength;
    int background;
} PD;

/* Include global functions */
#include "..\Larch\ProcessorCode.h"

/* Generate 8 bytes coded paramaters for FPGA */
PROCESSOR_API DWORD Generate(void *p, int *params, BYTE* coded)
{
    PD *pd = (PD*)p;
    int i,a;

    /* Zero out all values initially */
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        coded[i]=0;
    }

    /* Start Pixel */
    a = params[0] - 1;
    coded[0] = a & 255;
    coded[2] = a >> 8;

    /* Line Length */
    a = (params[0] - 1) + (params[1] - 1);
    coded[1] = a & 255;
    coded[2] = coded[2] | ((a >> 8) << 3);

    /* Minimum Pixel Value */
    coded[3] = params[2];

    /* Maximum Pixel Value */
    coded[4] = params[3];

    /* Synthesised Background */
    pd->background = params[4];
```

```
    pd->linelength = params[1];

    return pd->linelength;
}

/* Initialize processor data */
PROCESSOR_API int InitData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;
    pd->linelength = 2048;
    pd->background = 0;

    return sizeof(pbdata);
}

/* Initalize data before capture */
PROCESSOR_API void StartData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;

    return;
}

/* Decompress/Decode video data from camera */
PROCESSOR_API BOOL AddData(void *p, BYTE **indata, int *total, void *l)
{
    PD *pd = (PD*)p;
    lineinfo *li = (lineinfo*)l;
    int i=0,j;
    BYTE *data = *indata;
    BYTE mask[3] = {1,2,4};

    /* Look for sync bit */
    if (pd->synced == FALSE)
    {
        while (i<*total)
        {
            /* Skip flag byte */
            if ((i & 3) != 0)
            {
                /* a one in flag bit means compressed, a count of zero is a sync */
                if ( (( data[ i & 0xfffffffc ] & mask[ i & 3 ] ) != 0) &&
data[i]==0)
                {
                    pd->currentoffset = 0;
                    pd->synced = TRUE;
                    i++;
                    break;
                }
            }
            i++;
```

```
    }
}

/* Get current line */
li->line = pd->currentline;

/* Loop through all the new data */
while(i<*total)
{
    /* Go through a complete line */
    while ((pd->currentoffset < pd->linelength) && (i<*total))
    {
        /* Skip flag byte (only 3 bits) */
        if ((i & 3) != 0)
        {
            /* Flag bit is ? */
            if (( data[ i & 0xfffffffc ] & mask[ i & 3 ] ) == 0)
            {
                /* zero, not compressed */
                li->data[pd->currentoffset]=data[i];
                pd->currentoffset++;
            }
            else
            {
                /* one, compressed */
                for (j=0;j<data[i];j++)
                {
                    if (pd->currentoffset >= pd->linelength)
                    {
                        pd->currentoffset = pd->linelength - 1;
                    }
                    li->data[pd->currentoffset]=pd->background;
                    pd->currentoffset++;
                }
            }
        }
        i++;
    }
    /* End of line */
    if (pd->currentoffset >= pd->linelength)
    {
        /* Finished it properly */
        pd->synced = FALSE;
        pd->currentoffset = 0;
        li->line=pd->currentline;
        pd->currentline = pd->currentline + 1;

        /* Update data processed */
        i = i & 0xfffffffc;

        /* Update the line pointer */
        *indata += i;
        *total -= i;

        return TRUE;
    }
    else
    {
        /* Line not done */
```

```
            return FALSE;
        }
    }

    /* Data done, line not likely done */
    return FALSE;
}
```

D.9 Deltatracker Processor Code

D.9.1 Project file

D.9.1.1 Deltatracker.dsp

```
# Microsoft Developer Studio Project File - Name="Deltatracker" - Package
Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=Deltatracker - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Deltatracker.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Deltatracker.mak" CFG="Deltatracker - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Deltatracker - Win32 Release" (based on "Win32 (x86) Dynamic-Link
Library")
!MESSAGE "Deltatracker - Win32 Debug" (based on "Win32 (x86) Dynamic-Link
Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Deltatracker - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "DELTATRACKER_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS"
/D "_USRDLL" /D "DELTATRACKER_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
```

```
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386

!ELSEIF  "$(CFG)" == "Deltatracker - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "DELTATRACKER_EXPORTS" /Yu"stdafx.h" /FD /
GZ  /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
D "_MBCS" /D "_USRDLL" /D "PROCESSOR_EXPORTS" /FD /GZ /c
# SUBTRACT CPP /YX /Yc /Yu
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /out:"Debug/Deltatracker.vpr" /pdbtype:sept

!ENDIF

# Begin Target

# Name "Deltatracker - Win32 Release"
# Name "Deltatracker - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\Deltatracker.c
```

```
# End Source File
# Begin Source File

SOURCE=.\StdAfx.c
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\resource.h
# End Source File
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\Deltatracker.rc
# End Source File
# End Group
# End Target
# End Project
```

D.9.2 MFC Files

D.9.2.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
#define AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


// Insert your headers here
#define WIN32_LEAN_AND_MEAN// Exclude rarely-used stuff from Windows headers

#include <windows.h>
// #define _AFX_NOFORCE_LIBS
// #include "afxwin.h"

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.
```

```
#endif //
!defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
```

D.9.2.2 StdAfx.c

```
// stdafx.cpp : source file that includes just the standard includes
//  Lineup.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

D.9.3 Resource Files

D.9.3.1 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Lineup.rc
//
#define IDR_FPGA                        101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.9.3.2 Deltatracker.rc

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
```

```
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

/////////////////////////////////////////////////////////////////////////////
//
// FPGA
//

IDR_FPGA                FPGA    DISCARDABLE     "fpga.bin"

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

D.9.4 Deltatracker Code Files

D.9.4.1 Deltatracker.c

```
// Deltatracker.c : Deltatracker processor for second generation camera
//
```

```
#include "stdafx.h"

#include "resource.h"

/* Define video processor name */
#define PBNAME "Deltatracker Processor"
/* Number of parameters */
#define PBNUMOPT 5
/* Their defaults */
#define PBOPTDEFS 1,2048,6,1,128
/* Their titles */
#define PBOPT1 "Start Pixel"
#define PBOPT2 "Length of Line"
#define PBOPT3 "Threshold"
#define PBOPT4 "Bandwidth"
#define PBOPT5 "Synthesised Background"
/* The number of rules */
#define PBNUMRULES 10
/* The rules themselves */
#define PBRULES {0,2,1,0}, {0,3,0,8}, {1,2,1,0}, {1,3,0,8}, {2,2,1,0},
{2,3,31,0}, {3,2,0,0}, {3,3,15,0}, {4,2,0,0}, {4,3,255,0}

/* Set private data */
typedef struct _PD
{
    BOOL synced;
    int currentline;
    int currentoffset;
    int linelength;
    int background;
} PD;

/* Include global functions */
#include "..\Larch\ProcessorCode.h"

/* Generate 8 bytes coded paramaters for FPGA */
PROCESSOR_API DWORD Generate(void *p, int *params, BYTE* coded)
{
    PD *pd = (PD*)p;
    int i,a;

    /* Zero out all values initially */
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        coded[i]=0;
    }

    /* Start Pixel */
    a = params[0] - 1;
    coded[0] = a & 255;
    coded[2] = a >> 8;

    /* Line Length */
    a = (params[0] - 1) + (params[1] - 1);
    coded[1] = a & 255;
    coded[2] = coded[2] | ((a >> 8) << 3);

    /* Threshold */
```

```
    coded[3] = params[2];
    /* Bandwidth */
    coded[4] = params[3];

    /* Synthesised Background */
    pd->background = params[4];
    pd->linelength = params[1];

    return pd->linelength;
}

/* Initialize processor data */
PROCESSOR_API int InitData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;
    pd->linelength = 2048;
    pd->background = 0;

    return sizeof(pbdata);
}

/* Initalize data before capture */
PROCESSOR_API void StartData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;

    return;
}

/* Decompress/Decode video data from camera */
PROCESSOR_API BOOL AddData(void *p, BYTE **indata, int *total, void *l)
{
    PD *pd = (PD*)p;
    lineinfo *li = (lineinfo*)l;
    int i=0,j;
    BYTE *data = *indata;
    BYTE mask[3] = {1,2,4};

    /* Look for sync bit */
    if (pd->synced == FALSE)
    {
        while (i<*total)
        {
            /* Skip flag byte */
            if ((i & 3) != 0)
            {
                /* a one in flag bit means compressed, a count of zero is a sync */
                if ( (( data[ i & 0xfffffffc ] & mask[ i & 3 ] ) != 0) &&
data[i]==0)
                {
                    pd->currentoffset = 0;
```

```
                pd->synced = TRUE;
                i++;
                break;
            }
        }
        i++;
    }
}

/* Get current line */
li->line = pd->currentline;

/* Loop through all the new data */
while(i<*total)
{
    /* Go through a complete line */
    while ((pd->currentoffset < pd->linelength) && (i<*total))
    {
        /* Skip flag byte (only 3 bits) */
        if ((i & 3) != 3)
        {
            /* Flag bit is ? */
            if (( data[ i | 3 ] & mask[ i & 3 ] ) == 0)
            {
                /* zero, not compressed */
                li->data[pd->currentoffset]=data[i];
                pd->currentoffset++;
            }
            else
            {
                /* one, compressed */
                for (j=0;j<data[i];j++)
                {
                    if (pd->currentoffset >= pd->linelength)
                    {
                        pd->currentoffset = pd->linelength - 1;
                    }
                    li->data[pd->currentoffset]=pd->background;
                    pd->currentoffset++;
                }
            }
        }
        i++;
    }
    /* End of line */
    if (pd->currentoffset >= pd->linelength)
    {
        /* Finished it properly */
        pd->synced = FALSE;
        pd->currentoffset = 0;
        li->line=pd->currentline;
        pd->currentline = pd->currentline + 1;

        /* Update data processed */
        i = i & 0xfffffffc;

        /* Update the line pointer */
        *indata += i;
        *total -= i;
```

```
            return TRUE;
        }
        else
        {
            /* Line not done */
            return FALSE;
        }
    }

    /* Data done, line not likely done */
    return FALSE;
}
```

D.10 Fuzzy Logic Processor Code

D.10.1 Project file

D.10.1.1 Fuzzy.dsp

```
# Microsoft Developer Studio Project File - Name="Fuzzy" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=Fuzzy - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "Fuzzy.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "Fuzzy.mak" CFG="Fuzzy - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "Fuzzy - Win32 Release" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE "Fuzzy - Win32 Debug" (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Fuzzy - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "FUZZY_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "_MBCS"
/D "_USRDLL" /D "FUZZY_EXPORTS" /Yu"stdafx.h" /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "NDEBUG"
BSC32=bscmake.exe
```

```
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /machine:I386

!ELSEIF  "$(CFG)" == "Fuzzy - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "FUZZY_EXPORTS" /Yu"stdafx.h" /FD /GZ /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
D "_MBCS" /D "_USRDLL" /D "PROCESSOR_EXPORTS" /FD /GZ /c
# SUBTRACT CPP /YX /Yc /Yu
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
# ADD RSC /l 0x409 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /debug /machine:I386 /out:"Debug/Fuzzy.vpr" /pdbtype:sept

!ENDIF

# Begin Target

# Name "Fuzzy - Win32 Release"
# Name "Fuzzy - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"
# Begin Source File

SOURCE=.\Fuzzy.c
# End Source File
# Begin Source File

SOURCE=.\StdAfx.c
```

```
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\resource.h
# End Source File
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# Begin Source File

SOURCE=.\fpga.bin
# End Source File
# Begin Source File

SOURCE=.\Fuzzy.rc
# End Source File
# End Group
# End Target
# End Project
```

D.10.2 MFC Files

D.10.2.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
#define AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


// Insert your headers here
#define WIN32_LEAN_AND_MEAN// Exclude rarely-used stuff from Windows headers

#include <windows.h>
// #define _AFX_NOFORCE_LIBS
// #include "afxwin.h"

// TODO: reference additional headers your program requires here

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before
the previous line.
```

```
#endif //
!defined(AFX_STDAFX_H__A1FA25B3_40C9_11D2_A13A_0000C0059AB9__INCLUDED_)
```

D.10.2.2 StdAfx.c

```
// stdafx.cpp : source file that includes just the standard includes
//  Lineup.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

D.10.3 Resource Files

D.10.3.1 resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Lineup.rc
//
#define IDR_FPGA                        101

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.10.3.2 Fuzzy.rc

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
```

```
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

/////////////////////////////////////////////////////////////////////////////
//
// FPGA
//

IDR_FPGA                FPGA    DISCARDABLE     "fpga.bin"

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////



#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

D.10.4 Fuzzy Code Files

D.10.4.1 Fuzzy.c

```
// Fuzzy.c : Fuzzy processor for second generation camera
//
```

```
#include "stdafx.h"

#include "resource.h"

/* Define video processor name */
#define PBNAME "Fuzzy Processor 1024/5/3"
/* Number of parameters */
#define PBNUMOPT 4
/* Their defaults */
#define PBOPTDEFS 512,6,128,0
/* Their titles */
#define PBOPT1 "Staring Pixel"
#define PBOPT2 "Threshold"
#define PBOPT3 "Line average"
#define PBOPT4 "Jump adjustment"
/* The number of rules */
#define PBNUMRULES 8
/* The rules themselves */
#define PBRULES {0,0,0,0}, {0,3,0,4}, {1,2,2,0}, {1,3,32,0}, {2,2,32,0},
{2,1,223,0}, {3,2,0,0}, {3,3,15,0}

/* Set private data */
typedef struct _PD
{
    BOOL synced;
    int currentline;
    int currentoffset;
    int linelength;
} PD;

/* Include global functions */
#include "..\Larch\ProcessorCode.h"

/* Generate 8 bytes coded paramaters for FPGA */
PROCESSOR_API DWORD Generate(void *p, int *params, BYTE* coded)
{
    PD *pd = (PD*)p;
    int i;

    /* Zero out all values initially */
    for (i=0;i<GETPARAMS_MAX;i++)
    {
        coded[i]=0;
    }

    coded[0] = (params[0]-1) & 255;
    coded[1] = (params[0]-1) >> 8;
    coded[2] = params[1];
    coded[3] = max(params[2]-32,0);
    coded[4] = params[3];

    pd->linelength = 1024;

    return pd->linelength;
}

/* Initialize processor data */
PROCESSOR_API int InitData(void *p)
```

```
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;
    pd->linelength = 1024;

    return sizeof(pbdata);
}

/* Initalize data before capture */
PROCESSOR_API void StartData(void *p)
{
    PD *pd = (PD*)p;

    pd->synced = FALSE;
    pd->currentline = 0;
    pd->currentoffset = 0;

    return;
}

/* Decompress/Decode video data from camera */
PROCESSOR_API BOOL AddData(void *p, BYTE **indata, int *total, void *l)
{
    PD *pd = (PD*)p;
    lineinfo *li = (lineinfo*)l;
    int i=0,linebad=0;
    BYTE *data = *indata;

    /* Look for sync bit */
    if (pd->synced == FALSE)
    {
        while (i<*total)
        {
            if ((data[i] & 4))
            {
                pd->currentoffset = 0;
                pd->synced = TRUE;
                break;
            }
            i+=4;
        }
    }

    /* Get current line */
    pd->currentline = li->line;

    /* Loop through all the new data */
    while(i<*total)
    {
        /* Go through a complete line */
        while ((pd->currentoffset != pd->linelength) && (i<*total))
        {
            /* Skip flag byte (only 3 bits) */
            if ((i & 3) != 0)
            {
                li->data[pd->currentoffset]=data[i];
```

```
            pd->currentoffset++;
        }
        i++;
    }
    /* End of line */
    if (pd->currentoffset == pd->linelength)
    {
        /* Finished it properly */
        pd->synced = FALSE;
        pd->currentoffset = 0;

        /* Find the sync code */
        while (i<*total)
        {
            if ((i & 3) == 0)
            {
                if ((data[i] & 4))
                {
                    pd->synced = TRUE;
                    linebad = data[i] & 2;
                    break;
                }
            }
            i++;
        }

        /* Update data processed */
        *indata += i;
        *total -= i;

        /* Update the line pointer only if line defective */
        if (linebad)
        {
            pd->currentline = pd->currentline + 1;
        }
        li->line=pd->currentline;

        return TRUE;
    }
    else
    {
        /* Line not done */
        li->line=pd->currentline;
        return FALSE;
    }
}

/* Data done, line not likely done */
return FALSE;
}
```

Vita Auctoris

Roberto Muscedere (1973 Windsor, Ontario) received his B.A.Sc. degree from the University of Windsor, Windsor, Ontario. He received the Governors Medal for obtaining the highest academic average in his graduating class. In September 1996 he commenced his studies in the masters program at the University of Windsor and also took the position as the laboratory manager for the VLSI Research Group.

He is currently in the Ph.D. program in the VLSI Research Group at the University of Windsor where his area of research is in number systems and their VLSI implementation.