ERIC J. BOWDEN
MATT E. RICKS
IRENE K. THOMPSON

DIGITAL VIDEO RECORDER

COMPUTER ENGINEERING FINAL PROJECT
2006

DECEMBER 7, 2006

TABLE OF CONTENTS

INTRODUCTION AND MOTIVATION **2**

FUNCTIONAL SPECIFICATION **2**

HARDWARE FUNCTIONALITY 2
SOFTWARE FUNCTIONALITY 2

PROJECT IMPLEMENTATION **3**

DESIGN EVOLUTION 3
HARDWARE DESIGN 4
Power Supply *4*
Video Decoder *4*
Audio Decoder *4*
MPEG Encoder *4*
I^2C Interface *5*
SOFTWARE DESIGN 5
Middle-layer Software Interface *5*
Software-Hardware Interface 5
Stream Cast Server 6
Stream Cast Client 7
Media-Player *7*

CONCLUSIONS **10**

REFERENCES AND ACKNOWLEDGEMENTS **11**

APPENDIX A – BILL OF MATERIALS **12**

APPENDIX B – OMEGACORE SERVER CODE **16**

OmegaCoreServer.H *16*
OmegaCoreServer.Cpp *17*
StreamBuffer.h *31*
StreamBuffer.cpp *32*
Stdafx.h *34*
Stdafx.cpp *34*

APPENDIX C – I2C CONFIGURATION FILE **34**

APPENDIX D – MEDIA PLAYER CODE **35**

MediaPlayer.Java *35*
FrameGrabber.java *58*

APPENDIX E – BOARD PICTURES **67**

APPENDIX F – HARDWARE SCHEMATICS **68**

DVR Schematic *68*
PCB Design Layout *68*
PCB Top Layer *68*
PCB Bottom Layer *68*
PCB Top Layer Solder mask *68*
PCB Bottom Layer Solder Mask *68*
PCB Drill Points *68*
PCB Silk Screen *68*

LIST OF FIGURES AND TABLES

FIGURE 1 INITIAL HARDWARE BLOCK DIAGRAM..... 3

FIGURE 2 HARDWARE FUNCTIONALITY BLOCK DIAGRAM 4

FIGURE 3 HOST PC APPLICATION FLOW FOR THE HARDWARE-SOFTWARE INTERFACE AND THE STREAMING MEDIA SERVER INTERFACE. 5

FIGURE 4 HAND-SHAKE WAVEFORM USED TO COMMUNICATE BETWEEN THE USB INTERFACE AND THE MPEG ENCODER. IMAGE EXCERPTED FROM THE QUICKUSB USER'S GUIDE. 6

FIGURE 5 SCREENSHOT OF MEDIA PLAYER..... 8

FIGURE 6 COMMERCIAL DETECTION AND REPLACEMENT IN ACTION 9

INTRODUCTION AND MOTIVATION

In today's world using tape as a storage device has become obsolete. VCRs have made way for TIVO. Due to major developments in recent years in the way media is compressed and stored, we decided to explore these developments further for our senior project. We focused our attention on building a Digital Video Recorder. This device records a television program onto a hard disk drive, which allows the user to watch the recorded program at their convenience. With our product, the user is able to pause live television and rewind or fast-forward within a 60 minute timeframe.

This project allowed us to explore a variety of different fields of engineering, including analog signal processing, analog-to-digital conversion, digital storage and recall of encoded data in a real-time environment. This project proved to be a great learning experience in both hardware and software design.

FUNCTIONAL SPECIFICATION

HARDWARE FUNCTIONALITY

As a digital video recorder, the following features were designed and implemented:

- Receives an analog NTSC audio/video signal via an RCA cable.
- Converts the analog AV signal into a digital stream that conforms to the MPEG2 format.
- Communicates over USB between hardware and the host PC.

SOFTWARE FUNCTIONALITY

The software primarily stores and transmits the encoded data to clients, which, in turn display the MPEG in a custom media player.

- Stores the MPEG2 file to a host PC's hard drive.
- Server buffers and transmits the mpeg file to connected clients.
- Clients buffer as desired and forward data to the media player for decoding.
- Media player supports standard video manipulation operations like pause, rewind and fast-forward.
- Media player supports commercial detection and elimination.

Our DVR allows the user to store and view television programs on a PC. The user simply provides our device a NTSC television signal and interfaces the DVR hardware with a

personal computer through a USB2 port. A program installed on the PC will then display the program and give the user the ability to control the broadcast.

The media-player has two operating modes: live and playback. When in playback mode, a previously recorded program is displayed. When in live mode, an incoming MPEG stream is available to the user for viewing. In both modes of operation, the user has the ability to pause, rewind, and fast-forward the program. Furthermore, when in playback mode, an option is available which detects commercials. When this option is selected, commercial breaks are removed from the broadcast. The media-player developed for our product also has a slideshow button. When this feature is selected, a 30 second slideshow commences, while the broadcast continues in the lower right corner of the screen. The slideshow can be interrupted by the user at any time by moving the mouse. This feature is intended to allow the user to mask unwanted commercials with personal pictures.

PROJECT IMPLEMENTATION

DESIGN EVOLUTION

Our initial design involved fabricating a PCI card that would take a compound TV signal, tune to the desired channel, compress it into MPEG, and then store it on the hard drive. Subsequently, at the user's control, this data would be decoded, modulated and output to a television set. Our initial design can be found in figure 1.

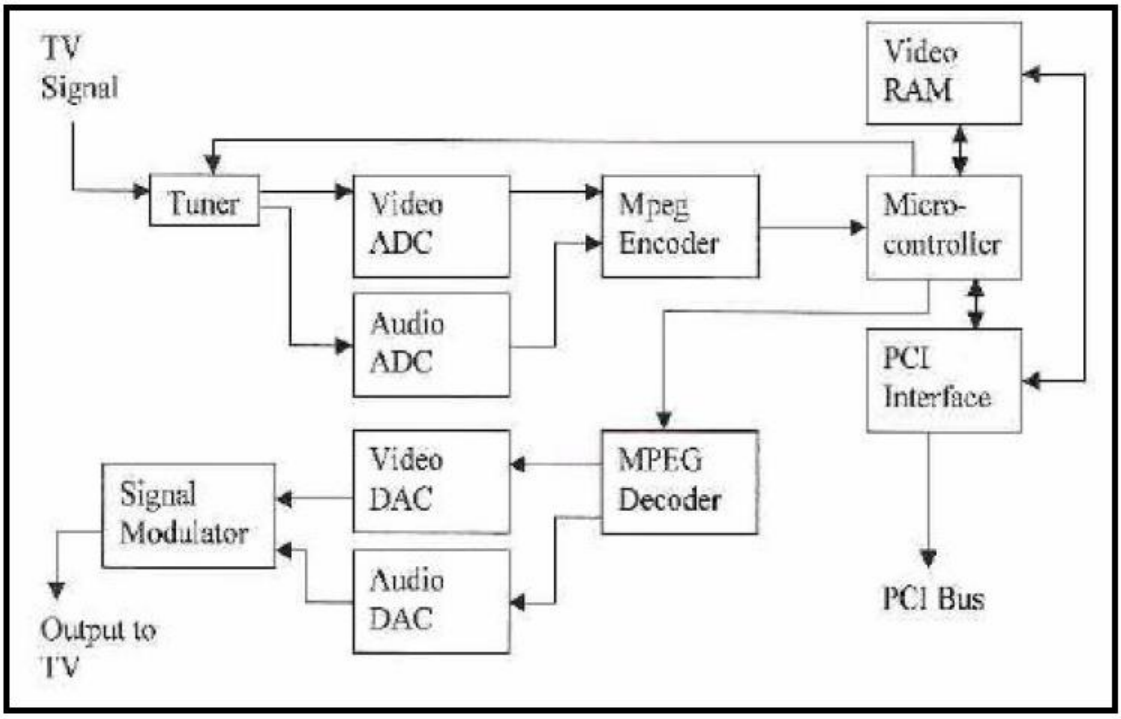


Figure 1 Initial hardware block diagram

Acquiring all the parts required for this design proved difficult. MPEG decoder chips were not available in quantities less than 1000 pieces. Furthermore, no tuner chips were available meeting our specifications. Due to these unforeseen difficulties we had to adapt our design. Our new design did not include a tuner and instead of displaying the program on a television set, it displays the MPEG stream on a PC media-player developed by our team. Additionally, to simplify the interface, we chose to use USB instead of PCI.

HARDWARE DESIGN

This section describes the hardware design and implementation. There are five main areas in this design; power supply, video decoder, audio decoder, mpeg encoder, and I^2C interface. A functional block diagram is shown below.

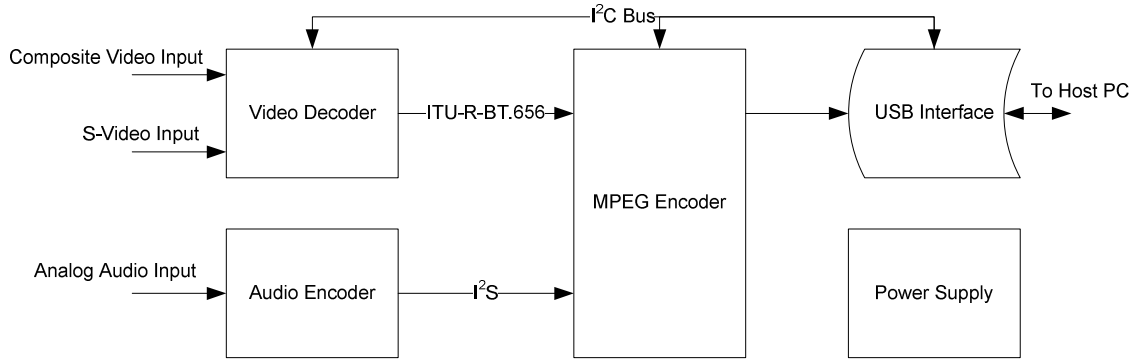


Figure 2 Hardware functionality block diagram

POWER SUPPLY

The chips on this board require two source voltages, 2.5 VDC and 3.3 VDC. The USB interface provides 5 VDC, but due to the power consumption of the MPEG Encoder chip, safety and feasibility considerations dictated the need for an external supply. To this extent, a DC power supply of 12 V is used, which is then down converted to 2.5 VDC and 3.3 VDC.

VIDEO DECODER

Philips SAA7114 is used for video decoding. It is a complete video input processor that takes care of all of the details of sync, filtering and color details. The data is converted into a digital stream (ITU-TR-BT.656 format compliant) that is sent to the MPEG encoder. The mode sync signals are embedded in the video data stream.

AUDIO DECODER

Philips DA1361 is used for analog-to-digital conversion of the audio input. The data is sent to the MPEG Decoder using I^2S.

MPEG ENCODER

The MPEG Encoder is the heart of this design. It is also a Philips chip (SAA6752). The video and audio inputs are compressed into MPEG-2 video and MPEG-1 layer 2 audio

streams and multiplexed into a single packet stream. The MPEG output stream is sent to the USB interface using the Data Expansion Bus Interface (DEBI) protocol.

I²C INTERFACE

The USB module is the I^2C bus master. It relays all I^2C commands from the host computer to the different components on the board.

SOFTWARE DESIGN

Our project has two separate software components. The first being the software interface between the USB module and the PC. This component is responsible for programming the Video Decoder and MPEG encoder chips. Furthermore, this module is responsible for collecting data from the MPEG encoder and storing it on the hard drive. The second software component is a media player. It allows the user to view and control a television program.

MIDDLE-LAYER SOFTWARE INTERFACE

The intermediate layer between the software media player and the encoder hardware consists of three main subsystems: a data extraction interface between the host pc and the USB hardware (and, by extension, the MPEG Encoder); a streaming media server that distributes the received data; and a client receiver that receives and buffers the data prior to passing it to the media player application. The first two subsystems are encapsulated in a single program that runs on the host pc to which the hardware module is attached. The receiving client may run on the same pc as the server, or on any machine that can access the host pc via TCP/IP. Figure 1 depicts the general program flow.

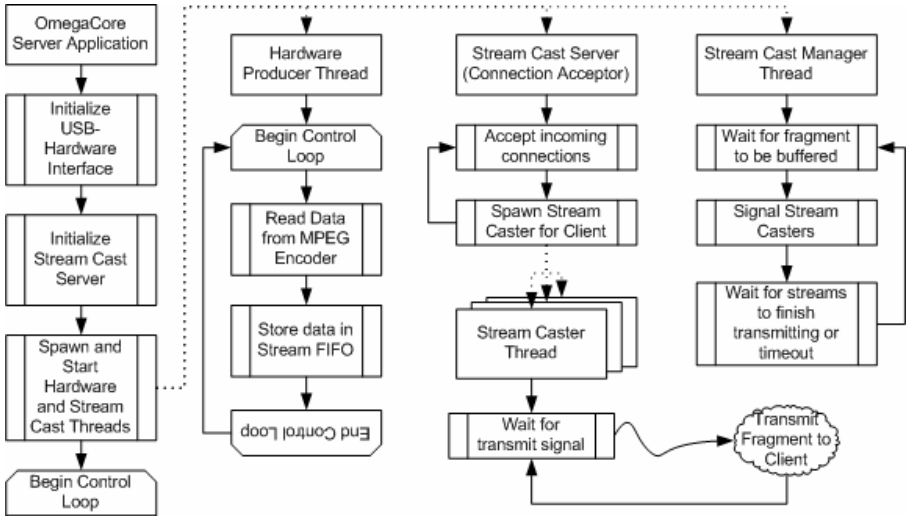


Figure 3 Host pc application flow for the hardware-software interface and the streaming media server interface.

Software-Hardware Interface

A prefabricated USB Interface chip, manufactured by QuickUSB, was used to communicate with the MPEG Encoder. The necessary low-level USB communication

software libraries were provided with the prefabricated board, speeding development. This USB interface provided two key functions: on-board chip configuration via I^2C, and data acquisition through a hardware-driven hand-shake protocol. The settings used for each chip may be seen in the appendix under *I2CInitializationScript.txt*.

After configuring the video decoder and MPEG encoder chips, the USB Interface chip was configured to perform full-handshake reads from the MPEG encoder. This involved loading the provided Full-Handshake firmware (quickusb-fullhs v2.11rc9.qusb) on to the QuickUSB chip. Connecting PB (Port B), RDYTST and READY on the USB chip to PDO, PDIDS and PDOVAL, respectively, on the MPEG Encoder, conforms to the DEBI handshake protocol standard. See Figure 2 for an example of this waveform. QuickUSB supports maximum data reads of 16MB using the asynchronous full handshake model; more than enough throughput for the intended application.

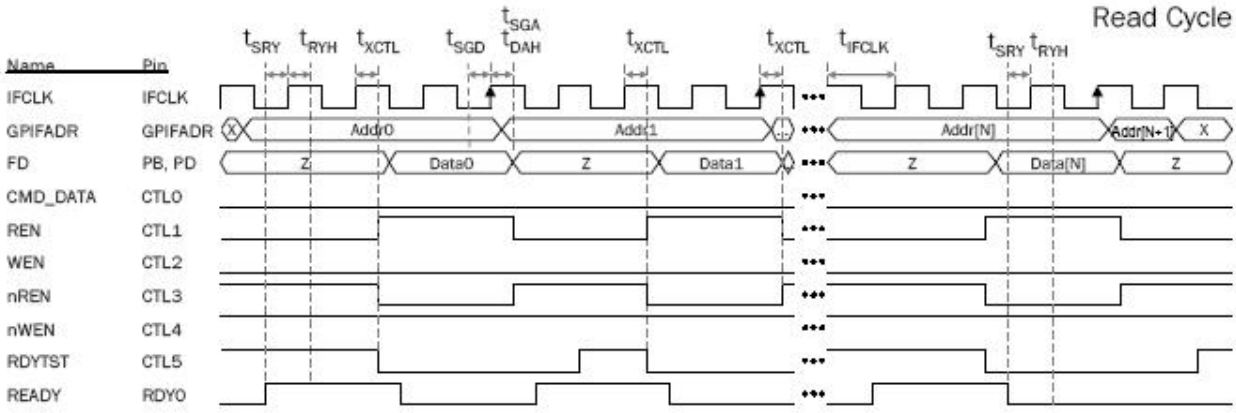


Figure 4 Hand-shake waveform used to communicate between the USB Interface and the MPEG Encoder. Image excerpted from the QuickUSB User's Guide.

Unfortunately, due to ill-documented error codes, the hardware data acquisition never worked reliably. Specifically, whenever a burst read command was initiated, in either synchronous or asynchronous mode, it summarily failed with a kernel-level IOCTL error. With only a generic error code, determining which particular interaction was the source of the failure became intractable. Attempts were made to use other handshaking methods, including manually toggling the pins (via general purpose I/O controls). The method, through actually producing data, proved too slow to generate a coherent MPEG on the byte level. All other handshaking methods involved hardware modifications—including adding more integrated circuit components—over-taxing the time-budget. From this point on, hardware data acquisition was simulated by reading from an MPEG file on the host pc's hard drive.

Stream Cast Server

The Stream Cast Server serves as the content distributor, implemented in C++, with SDL's network libraries. Clients connect over TCP/IP to the server, whereupon they are added to the list of clients to receive video. After the initial connection, all communication is one-way, from the server to the client (discounting heart-beats). The video data is stored

in an intermediate buffer as it is "generated." Once enough bytes have been received to qualify as a fragment (arbitrarily chosen as 256KB) the data is stored in a StreamBuffer (see appendix: StreamBuffer.h/c). Every time a fragment is buffered, a Stream Manager thread notifies all Stream Casters to send the latest fragment to their respective clients. Refer back to figure 1, and to the serverThread, streamCastManagerThread and streamCasterThread methods in the appendix ("OmegaCoreServer.h/c"). After the Stream Casters send their fragment (or timed-out), the fragment is completely discarded.

Stream Cast Client

The Java-implemented Stream Cast Client serves as the pre-processor and server-communication front-end, prior to sending the data to the media player. This stage serves as a buffer between the server and the media player. The input stream may then fork in two directions, directly to disk and then, indirectly, to the media player, or directly to the media player. This leverages Java's massive streaming-I/O infrastructure. At the present time however, the Java Media Framework, which the media player application was based on, does not support generic streams (but rather, RTP streams). Thus the indirect method of providing the data to the media player was used.

An important note, the whole scheme doesn't stream data in the modern sense of the word, since MPEG video does not periodically insert header information frames in the data. The lack of critical header/frame configuration data necessitates that all new client connections receive the header of the MPEG file immediately, followed by whatever fragment appears to be broadcasting currently. Another option, which mimics common commercial applications, would be to splice in metadata packets that indicate the status of the MPEG stream—those being stripped out and processed prior to sending the resulting stream to the MPEG decoder.

MEDIA-PLAYER

The media-player was required to display a MPEG stream, while giving the user the ability to control the broadcast. Furthermore, the media-player was expected to detect and remove commercials from a recorded broadcast. Finally, we wanted to give the user the ability to mask a commercial, while watching a live broadcast. When masking commercials, a slideshow of personal pictures displays for 30 seconds, while the broadcast continues in the lower-right corner of the screen.

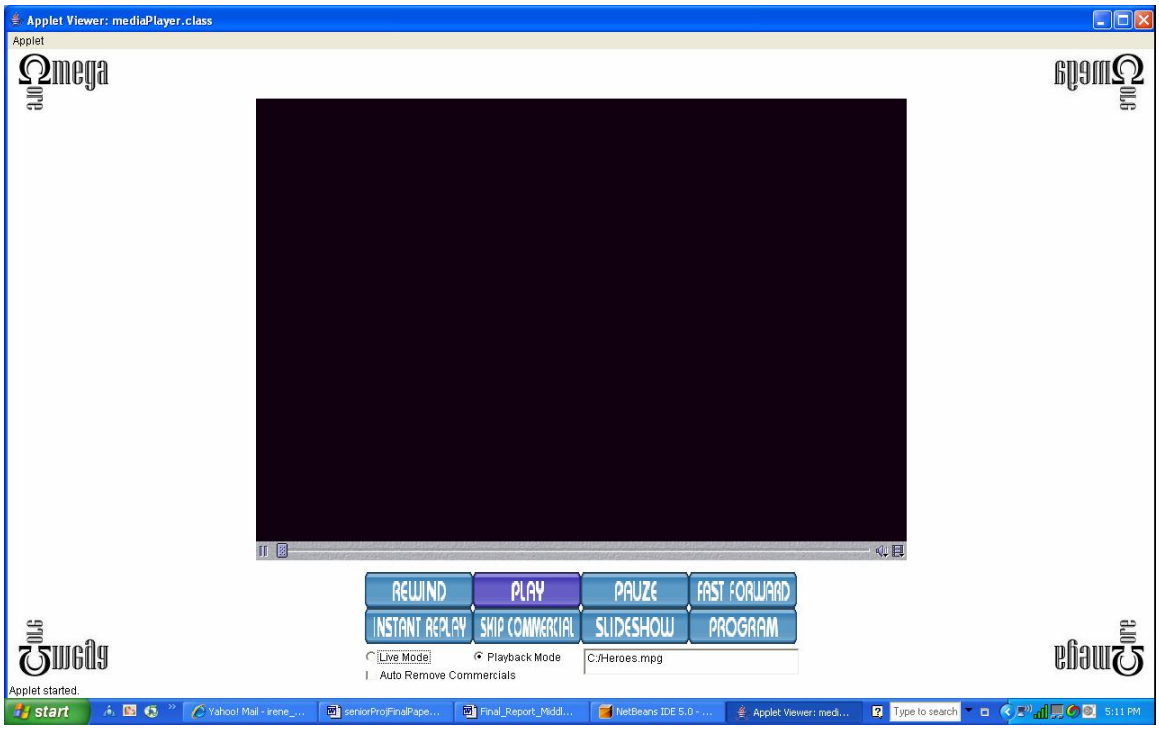


Figure 5 Screenshot of media player

The media-player for our project was developed in Java. It was built using Java Media Framework (JMF). JMF is an API that allows audio and video to be added to java applets. This optional package is available free for download at www.java.sun.com. Several controls were added to the player. The rewind, play, pause, and fastforward buttons have the expected functionality. When instant replay is selected by the user, the broadcast is set back 7 seconds in time. The skip commercial button jumps the broadcast forward by 30 seconds. A very important feature of the media-player is the slideshow functionality. When selected, a 30 second slideshow starts made up of personal pictures.

Figure 6 shows a screenshot of the media-player when the slideshow has been requested. The media-player program automatically extracts all the pictures stored in the user's MyPictures folder. The player will insert one new picture on the screen once every second. While the slideshow plays, the broadcast can be viewed in the lower right corner of the screen. If the user wants to end the slideshow and return to full screen broadcasting before the 30 seconds end, they simply have to move the mouse.

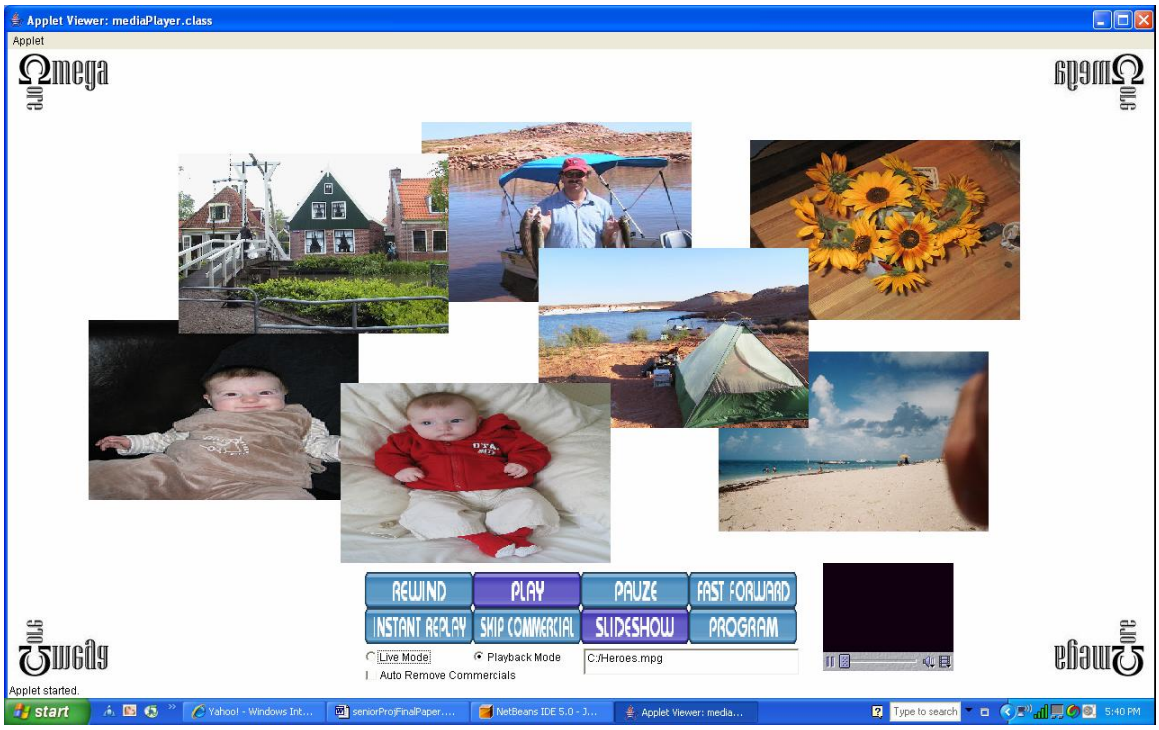


Figure 6 Commercial detection and replacement in action

The user also can select to run the player in either Live Mode or Playback Mode. When Playback Mode is chosen, the user has to enter the location of the file they wish to view in the applet. When the player is in live mode, it will display an incoming MPEG stream. Furthermore, there is the Auto Remove Commercials option available. This option will extract all commercials from a broadcast when in Playback Mode. These options are finalized and put in effect when the user presses the program button.

Commercials start and end with anywhere from 1 to 20 black frames. The commercial extraction functionality is composed of two parts. The first part identifies all the black frames in a media file. A text file is created identifying all the black frames and when they occur. This part is executed by a program called FrameGrabber. The media player analyzes the file created by the FrameGrabber program and determines where blocks of commercials start and finish. If the user has selected the Auto Remove Commercials option, the player will simply jump over commercial blocks when displaying the program.

The FrameGrabber program is based on a pass-through Video Codec available from Sun Developer Network called FrameAccess. The FrameGrabber program extracts all frames from a MPEG file and identifies the black frames. It groups all successive black frames and writes to a text file the number of black frames in each group and the location within the file this group of black frames was identified. This textfile is then passed on to the media-player. The media-player is only able to automatically remove commercials from a program when it is in playback mode.

When the media-player is in live mode, it has available to it a 1 hour, circular buffer to store the incoming stream. The user can fall 30 minutes behind the live stream and can buffer the last 30 minutes it has displayed. The player keeps track of how much of the broadcast has been buffered and how much buffered program is available beyond its current execution. That is, the player will ensure the user cannot fast-forward beyond where the incoming stream is being stored. The player also does not allow the user to fall behind more than 30 minutes behind the live broadcast. When the user tries to fall behind more than the allowed 30 minutes, by either pausing for an extended time or by rewinding the program, the player automatically goes into play mode to prevent the user from falling behind more than the allowed 30 minutes.

The code for the media-player and FrameGrabber can be found in Appendix D.

CONCLUSIONS

Even though we put a lot of effort in planning the project, there were many unexpected difficulties. In the original plan, we intended to implement an MPEG Decoder on the board, thus allowing exporting the MPEG stream to a television set. However, MPEG Decoder chips were not easily locatable, thus we chose to use a prefabricated PCI card (Vela CineView II) to do the MPEG decoding. In the process of writing the software to interface with this board, we stumbled across the ill-advertised requirement of a $1,200 SDK license. Since this requirement was only enforced at code run time, one week's worth of code had been written (and working) only to fail at the instantiation of a specific, required server module later in the development cycle (due to the lack of a server license), thus wasting time and money. Thus we chose to display the video via a custom made media player.

However, our biggest pitfall was choosing the QuickUSB module to facilitate communication between our hardware and the PC. We found solutions for all of our major and minor glitches, except this one. The product lacked the ability and documentation to successfully program it to communicate with our hardware using the required protocol, at the required speed. Specifically, a non-descript kernel-level IOCTL error prevented access to the data via the USB interface.

All in all this was a great learning experience. Given more time and resources we believe we could have overcome our interface problem by either building our own USB module or developing the DVR on a PCI card. Even though we could not see our hardware in action, we still feel the project was a success. This project allowed us to explore different aspects of engineering and allowed for a lot of creativity.

REFERENCES AND ACKNOWLEDGEMENTS

Abbott, Doug. 2004. PCI Bus Demystified, Second Edition. Newnes.

Ball, Stuart R. 2004. Analog Interfacing to Embedded Microprocessors, Second Edition. Newnes.

Encoder Reference Kit 12 Apr. 2004. UM10016_3 ISP1581 Hi-Speed USB MPEG2 Encoder Reference Kit. 7 Dec. 2006 <http://www.nxp.com/acrobat_download/usermanuals/UM10016_3.pdf >

Kovsky, Steve. High-Tech Toys for Your TV, Secrets of TiVo, Xbox, ReplayTV, UltimateTV and More. Que.

Pass-Through Video Codec 7 Dec. 2006. Accesing Individual Decoded Video Frames <http://java.sun.com/products/java-media/jmf/2.1.1/solutions/FrameAccess.html>

Simple Directmedia Layer. 7 Dec. 2006. Simple Directmedia Layer Main Page. <http://www.libsdl.org/index.php>.

Watkinson, John. 2004. The Mpeg Handbook. Focal Press.

Whitaker, Jerry. 2001. Television Receivers. McGraw-Hill.

Whitaker, Jerry. 2003. Video and Television Engineering, Fourth Edition. McGraw-Hill.

APPENDIX A – BILL OF MATERIALS

Item No.	*QTY*	*Manufacturer*	*Part No.*	*Description*	*Package Type*	*Source*	*Reference Designator*	*Cost Each*	*Total*
1	1	Philips	SAA7114	NTSC/PAL Digital Video Decoder	100-QFP	Digikey	U1	18.00	18.00
2	1	Philips	UDA1361	ADC Stereo Audio 24-bit	16-SSOP	Digikey	U2	3.42	3.42
3	1	Philips	SAA6752HS	Audio/Video Encoder MPEG	208-QFP	Digikey	U3	30.00	30.00
4	1	Micron	MT48LC4M16A2P-75	SDRAM 64MB 133MHz	54-TSOP	Digikey	U4	5.27	5.27
5	1	Maxim	MAX1626	DC/DC Converter 5V/3.3V	8-SOIC	Digikey	U5	4.61	4.61
6	2	On Semi	MMSF3P02HD	Mosfet p-channel 3A 20V	8-SOIC	Digikey	U6,U7	1.23	2.46
7	1	Maxim	MaAX1627	DC/DC Converter Adjustable	8-SOIC	Digikey	U8	4.61	4.61
8	1	Hirose	FX8-80S-SV	Connector Female 80 pin SMT		Digikey	J2	3.80	3.80
9	3	Rohm	MCR10EZHF18R0	Resistor 18 ohm	0805	Digikey	R2,R17,R18	0.04	0.12
10	3	Rohm	MCR10EZHF56R0	Resistor 56 ohm	0805	Digikey	R3,R16,R19	0.04	0.12
11	2	Rohm	MCR10EZHF10R0	Resistor 10 ohm	0805	Digikey	R6,R7	0.04	0.08
12	3	Vishay	CRCW08051R00FNEA	Resistor 1 ohm	0805	Digikey	R8,R11,R12	0.04	0.12
13	2	Rohm	MCR10EZHF4702	Resistor 47k	0805	Digikey	R9, R13	0.04	0.08

				ohm					
14	1	Rohm	MCR10EZHF47R0	Resistor 47 ohm	0805	Digikey	R10	0.04	0.04
15	3	Rohm	MCR10EZHF2201	Resistor 2.2k ohm	0805	Digikey	R14,R15,R27	0.04	0.12
16	5	Rohm	MCR10EZHF3301	Resistor 3.3k ohm	0805	Digikey	R20,R21,R22,R25,R26	0.04	0.20
17	2	Rohm	MCR10EZHF1002	Resistor 10k ohm	0805	Digikey	R23,R24	0.04	0.08
18	1	Rohm	MCR10EZHF1001	Resistor 1k ohm	0805	Digikey	R28	0.04	0.04
19	1	Rohm	MCR10EZHF2002	Resistor 20k ohm	0805	Digikey	R31	0.04	0.04
20	1	Rohm	MCR10EZHF2152	Resistor 21.5k ohm	0805	Digikey	R32	0.04	0.04
21	1	Rohm	MCR10EZHF2000	Resistor 200 ohm	0805	Digikey	R33	0.04	0.04
22	2	Vishay	WSL1206R0400FEA18	Resistor 0.4 ohm	0805	Digikey	R29,R30	1.45	2.90
23	2	Sumida	CDRH127NP-220MC	Power Inductor 22uH 3.6A	DR127	Digikey	L3,L4	2.91	5.82
24	1	Murata	BLM21AG121SN1D	Ferrite Chip 120 ohm 200 mA	0805	Digikey	L1	0.08	0.08
25	1	TDK	MLF2012E100K	Inductor 10uH	0805	Digikey	L2	0.39	0.39
26	5	Kemet	C0805C473K5RACTU	Capacitor 47nF	0805	Digikey	C1,C2,C3,C31,C33	0.06	0.30
27	43	Kemet	C0805C104K3RACTU	Capacitor 0.1uF	0805	Digikey	C4,C5,C6,C7,C8,C9,C10,C15,C20,C21,C22,C24,C26,C27,C28,C29,C30,C34,C35,C36,C37,C38,C39,C40,C41,C42,C43,C44,C45,C47,C48,C49,C50,C51,C52,C56,C	0.03	1.29

							61,C65,C66,C67,C68,C69,C70,C71		
28	1	Kemet	C0805C392K5RACTU	Capacitor 3.9nF	0805	Digikey	C9	0.05	0.05
29	1	Rohm	TCP0J106M8R	Capacitor 10uF, Tantalum	0805	Digikey	C11	0.36	0.36
30	4	Kemet	C0805C100J5GACTU	Capacitor 10pF	0805	Digikey	C12,C13,C53,C54	0.06	0.24
31	1	Kemet	C0805C102K5RACTU	Capacitor 1000pF	0805	Digikey	C14	0.04	0.04
32	1	Kemet	C0805C220J5GACTU	Capacitor 22pF	0805	Digikey	C46	0.10	0.10
33	2	Taiyo Yuden	EMK212BJ474KD-T	Capacitor 0.47uF	0805	Digikey	C57,C59	0.11	0.22
34	6	Rohm	TCA0J476M8R	Capacitor 47uF, Tantalum	1206	Digikey	C16,C17,C18,C19,C23,C25	0.33	1.98
35	2	Kemet	B45197A2227K509	Capacitor 220uF, Tantalum	7343-31	Digikey	C58,C60	1.18	2.36
36	1	Panasonic	ECJ-2VB2D221K	Capacitor 220pF	0805	Digikey	C62	0.07	0.07
37	1	Nichicon	UWT1E221MNL1GS	Capacitor 220uF, Electrolytic	G10	Digikey	C63	0.43	0.43
38	1	United Chemi-com	EMVA500ARA471MKG5S	Capacitor 470uF, Electrolytic	KG5	Digikey	C64	1.06	1.06
39	1	Abracon	ABLS-24.576MHZ-B2F-T	Crystal 24.576 Mhz	SMT	Digikey	X1	0.56	0.56
40	1	Abracon	ABLS-27.000MHZ-B2F-T	Crystal 27.000 Mhz	SMT	Digikey	X2	0.56	0.56
41	1	Stanley Electric	PG1111R-TR	LED Green	1206	Digikey	D4	0.27	0.27
42	1	Micro	1N5401-TP	Diode Rectifier	DO-201AD	Digikey	D1	0.15	0.15

43	2	Commercial On Semi	MBRS340T3G	3A 100V Diode Schottky	SMC	Digikey	D2,D3	0.38	0.76
44	1	Singatron Enterprises	2MJ-0002A110	Svideo Connector	PCB	Digikey	CON1	0.84	0.84
45	1	CUI Inc	PJ-102A	Power Jack Connector	PCB	Digikey	CON2	0.38	0.38
46	1	Various	AJ843G3	RCA Connector	PCB	Jameco	CON3	0.99	0.99
47	1	PCB123		DVR Printed Circuit Board		PCB123		131.49	131.49
48	1	Bitwise Systems	QUSB2	QuickUSB Module		Bitwise		149.00	149.00

APPENDIX B – OMEGACORE SERVER CODE

OMEGACORESERVER.H

```
#pragma once

#include "atlcoll.h"
#include "SDL_net.h"
#include "QuickUSB.h"
#include "StreamBuffer.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>

using namespace std;

struct StreamCastObject {
   HANDLE threadHandle;
   LPDWORD threadId;
   TCPsocket socket;
   int fragmentSize;
   char** fragmentVector;
   bool startTransmission;
   bool transmissionComplete;
   bool terminateStreamCast;
   int  interruptTransmission;
};

class OmegaCoreServer
{
public:
   OmegaCoreServer(void);
public:
   ~OmegaCoreServer(void);

   void printActiveConnections(void);

private:
   int loadConfigurationFile();
   void initializeNetworkInterface();
   int initializeDataRetrievalKernel();
   static DWORD WINAPI kernelThread(PVOID arg);
   static DWORD WINAPI serverThread(PVOID arg);
   static DWORD WINAPI streamCastManagerThread(PVOID arg);
   static DWORD WINAPI streamCasterThread(PVOID arg);

   void bufferByte(unsigned char byte);

   /* Hardware methods */
   int configureHardware();
```

```
   int resetHardware();
   int initializeUSB();
   bool isMPEGEncoderBusy();
   int closeUSB();
   int writeI2C(unsigned short mainAddress, unsigned char* subAddressData, unsigned short
length);
   int readI2C(unsigned short mainAddress, unsigned char* subAddressData, unsigned short
length);

   TCPsocket serverSocket;
   IPaddress serverIPAddress;
   int serverPort;

   HANDLE serverThreadHandle;
   HANDLE kernelThreadHandle;
   HANDLE streamCastManagerHandle;

   DWORD serverThreadId;
   DWORD kernelThreadId;
};
```

OMEGACORESERVER.CPP

```
/* Implementation of OmegaCoreServer Class
      Eric Bowden
      December 8, 2006 */
#include "StdAfx.h"
#include "OmegaCoreServer.h"

static bool shutdownFlag;  // Initiates shutdown for all threads (except the server
listener)
static bool shutdownServerListenerFlag; // Initiates shutdown for the server listener
static HANDLE usbHandle;  // Handle to access QuickUSB hardware
static int fragmentSize;  // Chunk size pertaining to both buffer increments and
transmission amounts
static StreamBuffer* streamBuffer;  // Buffer object that holds fragments
static char * currentStream; // The current fragment streaming out to the clients
static CAtlList<StreamCastObject*> serverConnectionList; // List of all active
connections to this server
static LPCTSTR lpszConnectionListName = (LPCTSTR)"OMEGACORE-CONNECTION-LIST-MUTEX"; //
Mutex designator
static char* simulationFile; // File path and name pertaining to the file to be
transmitted during simulation

#define PORTA (0x00) // QuickUSB Port A
#define PORTB (0x01) // QuickUSB Port B
#define PORTC (0x02) // QuickUSB Port C
#define PORTD (0x03) // QuickUSB Port D
#define PORTE (0x04) // QuickUSB Port E
#define KB (1024)
#define MB KB*KB

/* If SIMULATE is defined, the USB hardware will not be initialized and the server
   will simulate data retreival from the MPEG Encoder by reading and transmitting
   a file stored on disk. */
```

```
#define SIMULATE

// Supplemental method prototype
int axtoi(char *hexStg);  // Convert hexadecimal string to an int
char* intToIPString(int i, char * buff); // Convert a given 32-bit IP Address into a
string

OmegaCoreServer::OmegaCoreServer(void)
{
   cout << "BOOTING OMEGACORE STREAMCAST SERVER" << endl;
   serverPort = 9090;
   shutdownFlag = false;
   shutdownServerListenerFlag = false;
   fragmentSize = 1*MB;
   simulationFile = new char[2*KB];

   cout << "* Initializing Stream Buffer ..." << endl;
   streamBuffer = new StreamBuffer(fragmentSize, 20);
   cout << "... complete!" << endl;

   LPDWORD temp = NULL;

   // Create the streamer manager thread
   streamCastManagerHandle = CreateThread(NULL, 0, &streamCastManagerThread,
      NULL, CREATE_SUSPENDED, temp);

   cout << "* Processing configuration file ..." << endl;
   loadConfigurationFile();
   cout << "... complete!" << endl;

   cout << "* Initializing network interface ..." << endl;
   initializeNetworkInterface();
   cout << "... complete!" << endl;

   cout << "* Initializing hardware interface ..." << endl;
   initializeDataRetrievalKernel();
   cout << "... complete!" << endl;

   cout << "* Spawning StreamCast manager ..." << endl;
   ResumeThread(streamCastManagerHandle);
   cout << "... complete!" << endl;

   cout << "* Spawning connection listener ..." << endl;
   ResumeThread(serverThreadHandle);
   cout << "... complete!" << endl;

   cout << "* Spawning hardware data retriever ..." << endl;
   ResumeThread(kernelThreadHandle);
   cout << "... complete!" << endl;

   cout << "BOOT COMPLETE" << endl;
}

OmegaCoreServer::~OmegaCoreServer(void)
{
```

```
   cout << "Shutting down server ..." << endl;
   // Connection listener has to shut down first so no new clients are added to
   // the connection list.
   shutdownServerListenerFlag = true;
   cout << "Closing server connection listener ..." << endl;
   WaitForSingleObject(serverThreadHandle, INFINITE);
   shutdownFlag = true;
   cout << "Closing hardware data retrieval interface ..." << endl;
   WaitForSingleObject(kernelThreadHandle, INFINITE);

   cout << "Closing Stream Cast Manager ..." << endl;
   WaitForSingleObject(streamCastManagerHandle, INFINITE);

   delete streamBuffer;
   delete simulationFile;

   #ifndef SIMULATE
      resetHardware();
   /* Close USB Interface */
      closeUSB();
   #endif
}

void OmegaCoreServer::initializeNetworkInterface() {

   /* Initialize SDL Library */
   if(SDL_Init(0)==-1) {
      cerr << "SDL_Init: " << SDL_GetError() << endl;
       exit(1);
   }

   /* Initialize the networking portion of the SDL Library */
   if(SDLNet_Init()==-1) {
      cerr << "SDLNet_Init: " << SDLNet_GetError() << endl;
      exit(2);
   }

   /* Figure out who I am */
   if(SDLNet_ResolveHost(&serverIPAddress,NULL,serverPort) == -1) {
      cout << "SDLNet_ResolveHost: " << SDLNet_GetError() << endl;
      exit(3);
   }

   /* Open server socket */
   serverSocket = SDLNet_TCP_Open(&serverIPAddress);

   if(!serverSocket) {
      cerr << "SDLNet_TCP_Open: " << SDLNet_GetError() << endl;
      exit(4);
   }

   /* Create the server connection listener thread */
   serverThreadHandle = CreateThread(NULL, 0, &serverThread,
      &serverSocket, CREATE_SUSPENDED, &serverThreadId);
```

```
   if(serverThreadHandle == NULL) {
      cerr << "Failed to create server thread, returned error code: "
         << GetLastError() << endl;
      exit(5);
   }

}

/* Loads the custom configuration file that, as of now, only specifies the file
   to transmit if running in simulation mode. */
int OmegaCoreServer::loadConfigurationFile(void) {
   char* buffer = new char[2*KB];

   ifstream in("config.inf", ios_base::in);
   if(!in.is_open())
   {
      cerr << "Could not find config.inf" << endl;
      return -1;
   }
   while(!in.eof())
   {
      in.getline(buffer, 255);
      if(!strcmp("#SIMFILE", buffer)) {
         in.getline(buffer, 2*KB);
         strcpy_s(simulationFile, 2*KB, buffer);
      }
   }

   delete buffer;
   return 0;
}

/* This method is primarily hardware configuration.  QuickUSB is initialized,
   the board hardware is given a soft reset, and then configured.  The
   kernel thread (hardware communication thread) is also created here.*/
int OmegaCoreServer::initializeDataRetrievalKernel() {

   #ifndef SIMULATE
      initializeUSB();
      resetHardware();
      configureHardware();
   #endif

   /* Create kernel thread */
   kernelThreadHandle = CreateThread(NULL, 0, &kernelThread,
      NULL, CREATE_SUSPENDED, &kernelThreadId);

   if(kernelThreadHandle == NULL) {
      cerr << "Failed to create server thread, returned error code: "
         << GetLastError() << endl;
      exit(5);
   }

   return 0;
}
```

```
/* Method either retrieves data from the hardware, or from a file on disk.
   This data is buffered to the size of a fragment, and then placed in
   a fragment buffer in preparation for transmission. */
DWORD WINAPI OmegaCoreServer::kernelThread(PVOID arg) {
   const unsigned char* DATABYTE = new unsigned char[fragmentSize];
   unsigned long DATASIZE = fragmentSize;
   int result;

#ifndef SIMULATE
   unsigned short tempshort;
   ofstream outfile("C:/thuis.mpg", ios::out | ios::binary);

   Sleep(2000);
   result = QuickUsbReadSetting(usbHandle, 0x03, &tempshort);
   cout << "PostFIFO Config is: " << setbase(16) << tempshort << endl;

   result = QuickUsbSetTimeout(usbHandle, 3000UL);
   while(!shutdownFlag) {
      BYTE pid;
      result = QuickUsbReadDataAsync(usbHandle, const_cast<unsigned char
*>(DATABYTE),&DATASIZE, &pid);
      if(result==FALSE) {
         unsigned long error;
         QuickUsbGetLastError(&error);
         cout << "Failed to start Async - HSPP Error: " << error << endl;
      }
      result = QuickUsbAsyncWait(usbHandle, &DATASIZE, pid, 0);

      if(result != 0) {
         outfile.write((char *)(DATABYTE), INPUTBUFFERSIZE);
         outfile.flush();
      } else {
         unsigned long error;
         QuickUsbGetLastError(&error);
         cout << "HSPP Error: " << error << endl;
      }
   }

   outfile.flush();
#else
   ifstream input;
   input.open(simulationFile, ios::in | ios::binary);
   //input.open("C:/oldform.wmv", ios::in | ios::binary);
   if(!input.is_open()) {
      cout << "Input file failed to open ... " << endl << flush;
   }

   // Just keep sending the same file over and over
   while(!shutdownFlag) {
      while(!input.eof() && !shutdownFlag) { // && !(sendSuccess < 128)) {
         //cout << "Buffering a fragment ..." << endl;
         Sleep(10000);
         input.read((char *)(const_cast<unsigned char*>(DATABYTE)),DATASIZE);
```

```
         result = streamBuffer->bufferFragment((char *)(const_cast<unsigned
char*>(DATABYTE)));
         if(result == -1)
            cerr << "Uh oh, hit fragment limit." << endl;

      }
      input.close();
      input.open(simulationFile, ios::in | ios::binary);
   }
   input.close();
#endif
   delete DATABYTE;
   return 0;
}

/* Waits for incoming connections.  When one is detected, it creates a new
StreamCastObject
   and adds it to the list of active connections. */
DWORD WINAPI OmegaCoreServer::serverThread(PVOID arg)
{
   TCPsocket* serverSocket = static_cast<TCPsocket*>(arg);
   StreamCastObject* stream = NULL;
   HANDLE connectionListMutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE,
lpszConnectionListName);

   while(!shutdownServerListenerFlag) {
      stream = new StreamCastObject();

      stream->socket = SDLNet_TCP_Accept(*serverSocket);

      stream->fragmentSize = fragmentSize;
      stream->interruptTransmission = 0;
      stream->startTransmission = false;
      stream->terminateStreamCast = false;
      stream->transmissionComplete = false;
      stream->fragmentVector = &currentStream;

      if(stream->socket != NULL) {
         IPaddress* a = SDLNet_TCP_GetPeerAddress(stream->socket);
         int sendSuccess = 0;
         //cout << "Received client socket connection ... Host: " <<
         // intToIPString(a->host) << " Port: " << a->port << endl;

         stream->threadHandle = CreateThread(NULL, 0, &streamCasterThread,
            stream, CREATE_SUSPENDED, stream->threadId);

         WaitForSingleObject(connectionListMutex, INFINITE);
         serverConnectionList.AddTail(stream);
         ReleaseMutex(connectionListMutex);

         ResumeThread(stream->threadHandle);
      } else {
         delete stream;
      }
```

```
      stream = NULL;

      Sleep(500);
   }

   SDLNet_Quit();

   return 0;
}

/* Method monitors the data fragment buffer for new data to transmit.  The arg should
   be null.  When a new fragment is available, this method copies the pointer to the
   currentStream pointer and then signals to all the stream casters to start
transmitting.
   To terminate this thread, set the global shutdownFlag to true.  When terminating,
   this method with terminate all the streamCasters. */
DWORD WINAPI OmegaCoreServer::streamCastManagerThread(PVOID arg) {

   StreamCastObject* stream;
   bool allFinishedTransmitting;
   HANDLE connectionListMutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE,
lpszConnectionListName);

   while(!shutdownFlag) {
      //cout << "* SCM: Initial Spin-Wait" << endl;
      while(streamBuffer->numAllocatedFragments() == 0) {
         //cout << "Spinning ..." << endl;
         Sleep(50);
         if(shutdownFlag) break;
      }

      if(shutdownFlag) break;

      //cout << "Got a fragment! " << (int) streamBuffer->numAllocatedFragments() <<
endl;

      //cout << "* Retrieving allocated fragment" << endl;
      currentStream = streamBuffer->getAllocatedFragment();

      //cout << "* Starting transmissions" << endl;
      for(size_t i = 0; i < serverConnectionList.GetCount(); i++) {
         stream = serverConnectionList.GetAt(serverConnectionList.FindIndex(i));
         stream->startTransmission = true;
      }

      allFinishedTransmitting = false;

      //cout << "* Waiting for transmissions to complete" << endl;
      while(!allFinishedTransmitting) {
         allFinishedTransmitting = true;
         for(size_t i = 0; i < serverConnectionList.GetCount(); i++) {
            stream = serverConnectionList.GetAt(serverConnectionList.FindIndex(i));
            if(stream->transmissionComplete == false)
               allFinishedTransmitting = false;
         }
```

```
         if(shutdownFlag) break;

         Sleep(5);
      }

      //cout << "* Freeing fragment" << endl;
      streamBuffer->freeAllocatedFragment();
   }

   //cout << "* Sending signal to terminate streamers!" << endl;
   for(size_t i = 0; i < serverConnectionList.GetCount(); i++) {
      stream = serverConnectionList.GetAt(serverConnectionList.FindIndex(i));
      stream->terminateStreamCast = true;
   }

   //cout << "* Waiting for streamers to terminate ... " << endl;
   for(size_t i = 0; i < serverConnectionList.GetCount(); i++) {
      WaitForSingleObject( connectionListMutex, INFINITE );
      stream = serverConnectionList.RemoveHead();
      ReleaseMutex(connectionListMutex);

      WaitForSingleObject(stream->threadHandle, INFINITE);

      delete stream;
   }

   return 0;
}

/* The argument must be a pointer to the streamer's StreamCastObject.
   Thread that waits until a valid stream has been loaded in the currentThread
   (via the setting of a startTransmission boolean in the given thread's
   StreamCastObject.)  To cause the thread to terminate, set the
   terminateStreamCast boolean in the given thread's StreamCastObject to true. */
DWORD WINAPI OmegaCoreServer::streamCasterThread(PVOID arg) {

   StreamCastObject* sco = static_cast<StreamCastObject*>(arg);
   IPaddress* peerAddress = SDLNet_TCP_GetPeerAddress(sco->socket);
   int numBytesTransmitted = 0;

   //cout << "STREAM CASTER STARTED!" << endl;
   while(!sco->terminateStreamCast) {

      while(!sco->startTransmission) { Sleep(333); if(sco->terminateStreamCast) break;}
      if(sco->terminateStreamCast) break;
      //cout << "STARTING TRANSMISSION!" << endl;
      sco->startTransmission = false;

      numBytesTransmitted = SDLNet_TCP_Send(sco->socket, *(sco->fragmentVector), sco-
>fragmentSize);
      if(numBytesTransmitted < sco->fragmentSize) {
         cerr << "Client disconnected ..." << endl;
         sco->terminateStreamCast = true;
      }
```

```
      sco->transmissionComplete = true;
   }

   SDLNet_TCP_Close(sco->socket);
   return 0;
}

int OmegaCoreServer::initializeUSB() {
   char *namePtr;
   char nameList[120];
   int result;
   usbHandle = NULL;
   cout << "... Initializing USB Interface" << endl;

   //  Find the QuickUSB modules in the system
   QuickUsbFindModules(nameList, 128);

   // Check for no modules and bail if we don't find any
   if (*nameList == '\0') {
      printf("Couldn't find any modules\n");
      return -1;
   }

   // Print out the name of each module found
   namePtr = nameList;
   while (*namePtr != '\0') {
      printf("... Found %s\n", namePtr);
      namePtr = namePtr + strlen(namePtr);
   }

   // Open the first device
   result = QuickUsbOpen(&usbHandle, nameList);
   if (result == FALSE) {
      printf("Cannot open %s\n", nameList);
      return -1;
   }

   // Setup direction on I/O ports
   // 8-bit wide instead of 16
   // Splits Port B and Port D

   unsigned short tempshort = 0x00;

   // UNCOMMENT ME
   result = QuickUsbWritePortDir(usbHandle, PORTB, 0x00);  // PORTB all input
   if(result == FALSE) {
      cout << "Failed to port B direction to 0x00."  << endl;
   }

   result = QuickUsbWritePortDir(usbHandle, PORTD, 0x03);  // PORTD bit0,1 to out
   if(result == FALSE) {
      cout << "Failed to set port D direction to 0x03." << endl;
   }
   //UNCOMMENT ME
```

```
   result = QuickUsbWritePortDir(usbHandle, PORTC, 0x00);  // PORTC bit0 to in
   if(result == FALSE) {
      cout << "Failed to set port C direction to 0x00" << endl;
   }

   /* Set up high speed USB ME TOO*/
   result = QuickUsbReadSetting(usbHandle, 0x05, &tempshort);
   tempshort &= ~((3 << 3) | (1 << 15));
   tempshort |= (2 << 3) | (1 << 15);
   result = QuickUsbWriteSetting(usbHandle, 0x05, tempshort);
   if(result == FALSE) {
      cout << "Failed to write USB Setting Address 0x05" << endl;
   }
   unsigned short tempshort2;
   result = QuickUsbReadSetting(usbHandle, 0x05, &tempshort2);
   if(tempshort != tempshort2) {
      cout << "Writing USB Address 0x05 did not stick.  Want (" << setbase(16)
         << tempshort << "), got(" << setbase(16) << tempshort2 << ")." << endl;
   }

   result = QuickUsbReadSetting(usbHandle, 0x03, &tempshort);
   cout << "PreFIFO Config is: " << setbase(16) << tempshort << endl;
   tempshort &= ~0x03;
   //tempshort |= (1 << 1) | (1<<7) | (1 << 5);
   tempshort = 0xba;
   result = QuickUsbWriteSetting(usbHandle, 0x03, tempshort);
   if(result == FALSE) {
      cout << "Failed to enable GPIF pins." << endl;
   }
   result = QuickUsbReadSetting(usbHandle, 0x03, &tempshort);
   cout << "PostFIFO Config is: " << setbase(16) << tempshort << endl;

   unsigned short major, minor, build;
   QuickUsbGetDllVersion(&major, &minor, &build);
   cout << "DLL Version: " << setbase(10) << major << "." << minor << " build " << build
<< endl;
   QuickUsbGetDriverVersion(&major, &minor, &build);
   cout << "Driver Version: " << setbase(10) << major << "." << minor << " build " <<
build << endl;
   return 0;
}

int OmegaCoreServer::closeUSB() {
   if(usbHandle != NULL) {
      QuickUsbClose(usbHandle);
      return 0;
   }

   return 1;
}


/* Function reads in the I2C Configuration file "I2CInitializationScript.txt"
   that details various various commands and arguments that will be sent out on
   the I2C bus for chip initialization.  The commands are:
```

```
   #ADDR - the 8-BIT (!!!) address of the chip
   #WRITE . . . # - The first number represents the number of bytes (N) written to each
      of the following subaddress.
      Following which are (N+1)-tuples, where the first number is the subaddress and
      and the following N numbers are transmitted to the chip.
   */
int OmegaCoreServer::configureHardware() {
   unsigned short addr; // Holds the 7-bit chip address
   unsigned char subaddrdata[8]; // byte 0 - subaddress; bytes 1-7 - data
   unsigned char* subaddr = subaddrdata; // pointer to byte 0
   unsigned char* data = subaddrdata+1; // pointer to byte 1
   ifstream i2cinit("I2CInitializationScript.txt", ios_base::in); // I2C Script file
   char i2cdata[32]; // Buffer that holds lines from the script file
   unsigned char i2cverify[32]; // Buffer that holds data read from the I2C bus

   cout << "... Configuring hardware" << endl;
   /* This loop basically parses the script file in the following way, if line is:
         #I2CADDRESS, then read the next line which contains an 8-bit chip address

         #WRITE, the read the next line which contains the number of data bytes, N, that
            are to be sent to each of the subsequent subaddresses.  Then repeat the
            following until we see a line with a '#':
               Read next line (should contain an 8-bit subaddress)  Ignore blank lines
                  and lines that start with '%'
               Read next N lines (each line should contain a data byte) bytes are sent
                  in the order they are seen
               Send data to the chip defined by the addr and subaddr.
               Read the data back from the chip and make sure it matches what was sent
   */

   addr = 0x40>>1;
   subaddr[0] = 0x10;
   readI2C(addr, subaddrdata, 1);
   cout << "ENCODER MODE: " << setbase(16) << (short)subaddrdata[1] << endl;

   while(i2cinit.getline(i2cdata,16)) {
      if(!strcmp(i2cdata,"#I2CADDRESS")) {
         i2cinit.getline(i2cdata,128);  // get base addr
         addr = ((short)axtoi(i2cdata)>>1); // Make it a 7-bit addr
         cout << setw(4) << " " << "I2C Base address: " << setbase(16) << addr << endl;
      } else if(!strcmp(i2cdata,"#WRITE")) {
         i2cinit.getline(i2cdata,128); // Get number of bytes per write
         int count = axtoi(i2cdata);

         i2cinit.getline(i2cdata,128);  // Get either subaddress, comment or end of write
block
         while(i2cdata[0] != '#') {
            if(i2cdata[0] != NULL && i2cdata[0] != '%') {
               subaddr[0] = (char)axtoi(i2cdata);
               for(int i = 0; i < count; ++i) {
                  i2cinit.getline(i2cdata,128); // Get data to write
                  data[i] = (char)axtoi(i2cdata);
               }
               if(addr == (0x40>>1)) {
                  while(isMPEGEncoderBusy()) {
```

```
                    cout << setw(8) << " " << "MPEG Encoder Busy" << endl;
                    Sleep(500);
                }
            }
            writeI2C(addr, subaddrdata, count);
            cout << setw(8) << " " << "Sub-address: " << setbase(16) <<
(short)subaddr[0]
                 << " data[0]: " << setbase(16) << (short)data[0] << endl;
            /* Verify correct data written */
            i2cverify[0] = subaddr[0];
            readI2C(addr, &i2cverify[0], count);
            for(int i = 1; i <= count; i++) {
               if(i2cverify[i] != subaddrdata[i]) {
                  cout << "write/read inconsistancy: addr: " << setbase(16) << addr
                     << " subaddress: " << setbase(16) << (short)subaddrdata[0]
                     << " byte: " << count
                     << " written: " << setbase(16) << (short)subaddrdata[i]
                     << " read: " << setbase(16)   << (short)i2cverify[i]
                     << endl;
               }
            }
         }

         i2cinit.getline(i2cdata,128);
      }
   } else if(i2cdata[0] == NULL) {
      //Do Nothing for blank lines
   } else {
      cout << "Malformed line in script file ... (" << i2cdata << ")" << endl;
   }
}

addr = 0x40>>1;
subaddr[0] = 0x02;
data[0] = 0xFF;
writeI2C(addr, subaddrdata, 0);

subaddr[0] = 0x10;
readI2C(addr, subaddrdata, 1);
cout << "ENCODER MODE: " << setbase(16) << (short)subaddrdata[1] << endl;
return 0;
}

/* Initiates bit-banging sequence that resets the chips on the board. */
int OmegaCoreServer::resetHardware() {
   unsigned char data[1];
   int result;

   data[0] = 0x02;
   result = QuickUsbWritePort(usbHandle, PORTD, &data[0], 1);
   if(result == FALSE) {
      cout << "Setting reset pin low failed!" << endl;
   }
   else
   {
```

```
      // Do nothing
   }
   Sleep(200);
   data[0] = 0x03;
   result = QuickUsbWritePort(usbHandle, PORTD, &data[0], 1);
   if(result == FALSE) {
      cout << "Setting reset pin high failed!" << endl;
   }
   else
   {
      // Do nothing
   }
   return 0;
}

/* Returns whether the MPEG Encoder chip is busy (and thus is ignoring I2C commands. */
bool OmegaCoreServer::isMPEGEncoderBusy()
{
   unsigned short addr = 0x40>>1;   // Holds the 7-bit chip address
   unsigned char subaddrdata[2]; // byte 0 - subaddress; bytes 1-7 - data
   unsigned char* subaddr = subaddrdata; // pointer to byte 0
   unsigned char* data = subaddrdata+1; // pointer to byte 1
   subaddr[0] = 0x10;

   readI2C(addr, subaddrdata, 1);
   if(data[0] & (1 << 5))
      return true;
   else
      return false;
}

/* Wrapper that facilitates the proper sequence of I2C commands for writing to said bus.
   mainaddress - the 7-bit address of the chip
   subAddressData - arbitrary length where the first byte is the subaddress
         and all subsequent bytes are the payload to be transmitted.
   length - length of the PAYLOAD ONLY */
int OmegaCoreServer::writeI2C(unsigned short mainAddress, unsigned char* subAddressData,
unsigned short length) {
   int result;
   result = QuickUsbWriteI2C(usbHandle, mainAddress, subAddressData, length+1);
   if(result == FALSE) {
      cerr << "Failed to write I2C!" << endl;
   }
   return result;
}


/* Wrapper that facilitates the proper sequence of I2C commands for reading.
   mainaddress - the 7-bit address of the chip
   subAddressData - arbitrary length where the first byte is the subaddress
         and all subsequent bytes are the payload.  The payload will be overwritten
         by data from the bus.
   length - length of the PAYLOAD ONLY */
int OmegaCoreServer::readI2C(unsigned short mainAddress, unsigned char* subAddressData,
unsigned short length) {
```

```
   int result;
   QuickUsbWriteI2C(usbHandle, mainAddress, subAddressData, 1);
   result = QuickUsbReadI2C(usbHandle, mainAddress, subAddressData+1,&length);
   return result;
}

/* Prints a list of the current connections to stdout. */
void OmegaCoreServer::printActiveConnections(void) {
   HANDLE connectionListMutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE,
lpszConnectionListName);
   StreamCastObject* stream; // Temporary stream
   IPaddress* a; // Temporary IP Address for deriving host ip and port
   size_t numCon = serverConnectionList.GetCount(); // Number of active connections
   char buff[16];

   cout << "Active Connections:" << endl;

   if(numCon == 0) {
      cout << "No active connections." << endl;
      return;
   }

   WaitForSingleObject(connectionListMutex, INFINITE);
   for(size_t i = 0; i < numCon; i++) {
      stream = serverConnectionList.GetAt(serverConnectionList.FindIndex(i));
      if(stream->terminateStreamCast == false) {
         a = SDLNet_TCP_GetPeerAddress(stream->socket);
         cout << "[" << i << "] Host: " << intToIPString(a->host, buff) << " Port: " <<
a->port << endl;
      }
   }
   ReleaseMutex(connectionListMutex);
}
/* Supplemental method used in configureHardware to convert an ASCII string
   of hexadecimal values to an int. */
int axtoi(char *hexStg) {
  int n = 0;          // position in string
  int m = 0;          // position in digit[] to shift
  int count;          // loop index
  int intValue = 0;   // integer value of hex string
  int digit[5];       // hold values to convert
  while (n < 4) {
     if (hexStg[n]=='\0')
        break;
     if (hexStg[n] > 0x29 && hexStg[n] < 0x40 ) //if 0 to 9
        digit[n] = hexStg[n] & 0x0f;            //convert to int
     else if (hexStg[n] >='a' && hexStg[n] <= 'f') //if a to f
        digit[n] = (hexStg[n] & 0x0f) + 9;      //convert to int
     else if (hexStg[n] >='A' && hexStg[n] <= 'F') //if A to F
        digit[n] = (hexStg[n] & 0x0f) + 9;      //convert to int
     else break;
    n++;
  }
  count = n;
  m = n - 1;
```

```
  n = 0;
  while(n < count) {
     // digit[n] is value of hex digit at position n
     // (m << 2) is the number of positions to shift
     // OR the bits into return value
     intValue = intValue | (digit[n] << (m << 2));
     m--;   // adjust the position to set
     n++;   // next digit to process
  }
  return (intValue);
}


/* Custom method that turns a 32-bit (reverse order)
   ip address into a string. */
char* intToIPString(int i, char * buff) {
   char* PIP = buff;
   int mask = 0xFF;
   _itoa_s((i)&mask,PIP,4,10);
   PIP += strlen(PIP)+1;
   PIP[-1] = '.';
   _itoa_s((i >> 8)&mask,PIP,4,10);
   PIP += strlen(PIP)+1;
   PIP[-1] = '.';
   _itoa_s((i >> 16) &mask,PIP,4,10);
   PIP += strlen(PIP)+1;
   PIP[-1] = '.';
   _itoa_s((i>>24)&mask,PIP,4,10);
   buff[15] = NULL;

   return buff;
}
```

STREAMBUFFER.H

```
#pragma once

#include "atlcoll.h"
#include <windows.h>

class StreamBuffer {
public:
   StreamBuffer(int fragSize, int numFragments);
   ~StreamBuffer();

   int bufferFragment(char* partialFrag, int numBytes);
   int bufferFragment(char* frag);
   size_t numAllocatedFragments();
   char* getAllocatedFragment();
   int freeAllocatedFragment();

private:

   int fragmentSize;
   int fragmentCap;
```

```
   char* currentFragment;
   int fragLevel;
   CAtlList<char*> freeFragments;
   CAtlList<char*> allocatedFragments;
};
```

STREAMBUFFER.CPP

```
/*  Eric J. Bowden
   December 8, 2006

   StreamBuffer.cpp
   Data structure (FIFO) that holds "fragments" of data of a settable size.
   This library is threadsafe. */

#include "StdAfx.h"
#include "StreamBuffer.h"
#include "assert.h"

LPCTSTR bufferMutex = (LPCTSTR)"OMEGACORE-STREAMBUFFER-MUTEX";
HANDLE bufferMutexHandle;

StreamBuffer::StreamBuffer(int fragSize, int numFragments) {
   char* tempFragment;
   fragmentSize = fragSize;
   fragmentCap = numFragments;
   fragLevel = 0;

   for(int i = 0; i < fragmentCap; i++) {
      tempFragment = new char[fragmentSize];
      freeFragments.AddHead(tempFragment);
   }

   currentFragment = freeFragments.GetHead();

   bufferMutexHandle = OpenMutex(MUTEX_ALL_ACCESS, NULL, bufferMutex);
}

StreamBuffer::~StreamBuffer() {
   size_t size = freeFragments.GetCount();
   char* tempFragment;

   for(size_t i = 0; i < size; i++) {
      tempFragment = freeFragments.RemoveHead();
      delete tempFragment;
   }

   size = allocatedFragments.GetCount();
   for(size_t i = 0; i < size; i++) {
      tempFragment = allocatedFragments.RemoveHead();
      delete tempFragment;
   }
}

int StreamBuffer::bufferFragment(char* partialFrag, int numBytes) {
```

```
   int bytesLeft;
   int amtToBuffer;
   int partialFragOffset = 0;

   if(freeFragments.GetCount() == 0) {
      currentFragment = NULL;
      return -1;
   }

   assert(currentFragment == freeFragments.GetHead());

   do {
      bytesLeft   = fragmentSize - fragLevel;
      amtToBuffer = min(bytesLeft, numBytes);
      numBytes -= amtToBuffer;

      memcpy_s(currentFragment+fragLevel, fragmentSize, partialFrag+partialFragOffset,
amtToBuffer);
      partialFragOffset += amtToBuffer;
      fragLevel         += amtToBuffer;

      if(fragLevel >=  fragmentSize) {
         WaitForSingleObject(bufferMutexHandle, INFINITE);
         allocatedFragments.AddTail(freeFragments.RemoveHead());
         ReleaseMutex(bufferMutexHandle);
         fragLevel = 0;
         if(freeFragments.GetCount() == 0) {
            currentFragment = NULL;
            return -1;
         } else {
            WaitForSingleObject(bufferMutexHandle, INFINITE);
            currentFragment = freeFragments.GetHead();
            ReleaseMutex(bufferMutexHandle);
         }
      }

   } while(numBytes != 0);

   return 0;
}

int StreamBuffer::bufferFragment(char* frag) {
   return bufferFragment(frag, fragmentSize);
}

size_t StreamBuffer::numAllocatedFragments() {
   return allocatedFragments.GetCount();
}
char* StreamBuffer::getAllocatedFragment() {
   if(allocatedFragments.GetCount() == 0) return NULL;

   return allocatedFragments.GetHead();
}

int StreamBuffer::freeAllocatedFragment() {
```

```
   assert(allocatedFragments.GetCount() > 0);
   WaitForSingleObject(bufferMutexHandle, INFINITE);
   freeFragments.AddTail(allocatedFragments.RemoveHead());
   ReleaseMutex(bufferMutexHandle);
   if(freeFragments.GetCount() == 1) {
      WaitForSingleObject(bufferMutexHandle, INFINITE);
      currentFragment = freeFragments.GetHead();
      ReleaseMutex(bufferMutexHandle);
   }
   return 0;
}
```

STDAFX.H

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once


#define WIN32_LEAN_AND_MEAN      // Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>
```

STDAFX.CPP

```
// stdafx.cpp : source file that includes just the standard includes
// OmegaCoreServerApplication.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

   #include "stdafx.h"
```

APPENDIX C – I2C CONFIGURATION FILE

Columnized to save space.

```
#I2CADDRE       04              09              0e              13              18
SS              90              40              89              00              40
42
#WRITE          05              0a              0f              14              19
1               90              80              a4              00              80

01              06              0b              10              15              40
8               eb              44              0e              11              40

02              07              0c              11              16              41
C0              e0              40              00              fe              ff

03              08              0d              12              17              42
10              98              00              00              40              ff
```

```
43                  45                  9b                  00
ff          57      87          ac      00          be      00
44                  01                  9c                  00
ff          58      88          ad      02          bf      00
45                  f0                  9d                  #
ff          59      90          ae      00          #I2CADDRE
46                  00                  9e                  SS
ff          5a      91          f0      b0          40      #WRITE
47                  08                  9f                  1
ff          5b      92          00      b1          04      30
48                  10                  a0                  03
ff          5c      93          01      b2          00      32
49                  00                  a1                  02
ff          5d      94          00      b3          04      33
50                  10                  a2                  01
ff          5E      95          00      b4          00      40
51                  00                  a4                  01
ff          5F      96          80      b8          00      42
52                  d0                  a5                  01
ff          80      97          40      b9          00      d0
53                  02                  a6                  00
ff          83      98          40      ba          00      #WRITE
54                  0a                  a8                  2
5f          84      99          00      bb          00      %b1
55                  00                  a9                  %07
ff          85      9a          04      bc          00      %d0
56                  f2                  aa                  #
ff          86                  00          bd
```

APPENDIX D – MEDIA PLAYER CODE

MEDIAPLAYER.JAVA

```
/*
 * mediaPlayer.java
 *
 *
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
```

```
 */

/**
 *
 * This media player can display MPEG and many other media files.
 * It provides the user with standard controls such as play, ff,
 * rewind, etc. It also skips over commercial breaks when this
 * option has been selected by the user. Finally, it can generate
 * a 30 second slideshow based on the pictures found in the user's
 * MyPictures folder.
 */
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
import javax.media.Time;
import javax.media.*;
import java.util.Date;
import java.util.Vector;
import java.io.*;
import java.awt.PopupMenu;
import OmegaCore.OmegaCoreReceiver;
import javax.media.rtp.OutputDataStream;
import javax.media.rtp.RTPPushDataSource;

public class mediaPlayer extends Applet implements ActionListener, MouseListener,
Runnable { //implements MouseListener {

    /** Initialization method that will be called after the applet is loaded
     *  into the browser.
     */
    //MediaPlayer mediaPlayer1;
    //SymComponent aSymComponent;


    /* Buttons used in the applet */
    Button PlayButton;
    Button RewindButton;
    Button PauzeButton;
    Button FastForwardButton;
    Button InstantReplayButton;
    Button Skip30Button;
    Button Mask30Button;
    CheckboxGroup modeSelection;
    Checkbox liveMode;
    Checkbox playbackMode;
    TextField fileName;
    Checkbox commercialsGone;
    int xpos;
    int ypos;

    /* booleans used to keep track of the function requested by the user */
    boolean RewindButtonPressed, PlayButtonPressed, PauzeButtonPressed,
FastForwardButtonPressed;
    boolean InstantReplayButtonPressed, Skip30ButtonPressed, Mask30ButtonPressed;
    Thread rewindThread;
```

```
Thread slideShowThread;
boolean running = true;
boolean rewindReq = false;
boolean startMask = false;
boolean endMask = false;
double endMaskTime;
Date date;
CaptureDeviceInfo cdi;
//Time time;

/* Arrays of images used by the slideshow feature */
Image[] images = new Image[6];
Image[] images2 = new Image[6];
Image[] images3 = new Image[6];
Image[] images4 = new Image[6];
Image[] images5 = new Image[4];
Image[] images6 = new Image[4];
Image[] images7 = new Image[4];
Image[] playArray = new Image[2];
Image[] rewindArray = new Image[2];
Image[] pauzeArray = new Image[2];
Image[] fastforwardArray = new Image[2];
Image[] instantreplayArray = new Image[2];
Image[] slideshowArray = new Image[2];
Image[] skipcommercialArray = new Image[2];
Image[] programArray = new Image[2];
Image[] randomImages = new Image[28];

int imageIndex = 0;
int imageIndex2 = 0;
int imageIndex3 = 0;
int imageIndex4 = 0;
int imageIndex5 = 0;
int imageIndex6 = 0;
int imageIndex7 = 0;

/* indexes used by the image Buttons */
int playArrayIndex = 1;
int fastforwardArrayIndex = 0;
int rewindArrayIndex = 0;
int pauzeArrayIndex = 0;
int instantreplayArrayIndex = 0;
int slideshowArrayIndex = 0;
int skipcommercialArrayIndex = 0;
int programArrayIndex = 0;

Graphics Q;
Graphics bufferGraphics;
Image offscreen;
Dimension dim;
boolean slideShowActive = false;
int slideShowCounter = 2;
int picturesShown = 0;
Image fancyPlay;
Image testImage;
```

```
    /* variables used to contain the buffers in live mode */
    double bufferedAhead = 0.0;
    double bufferedBehind = 0.0;
    String lastCommand = "Play";
    double lastPlayCheck = 0.0;
    double pauzeStartTime = 0.0;
    double ffStartTime = 0.0;


    Button t;

    double[] commercialArray = new double[14];

    // Defaults settings for the player
    String playingMode = "playback";
    String playbackFile = "C:/Heroes.mpg";
    String commercialDetectionFile = "C:/Heroes.txt";

    String lastFile = "C:/Heroes.mpg";

    boolean commercialsGoneBoolean = false;

    int rewindNumber = 1;

    public void init() {

        // get all the images from the My Pictures file
        getImagesFromFile(new java.lang.String("C:/Documents and Settings/Irene T/My
Documents/My Pictures"),
                randomImages, randomImages.length);


        // set up the image arrays used in the slideshow feature
        images[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images[1] = randomImages[0];
        images[2] = randomImages[1];
        images[3] = randomImages[2];
        fancyPlay = getImage(getDocumentBase(), "file:///C:/playbutton.png");

        images2[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images2[1] = randomImages[3];
        images2[2] = randomImages[4];
        images2[3] = randomImages[5];

        images3[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images3[1] = randomImages[6];
        images3[2] = randomImages[7];
        images3[3] = randomImages[8];


        images4[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images4[1] = randomImages[9];
        images4[2] = randomImages[10];
        images4[3] = randomImages[11];
```

```
        images5[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images5[1] = randomImages[12];
        images5[2] = randomImages[13];
        images5[3] = randomImages[14];

        images6[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images6[1] = randomImages[15];
        images6[2] = randomImages[16];
        images6[3] = randomImages[17];

        images7[0] = getImage(getDocumentBase(), "file:///C:/white.jpg");
        images7[1] = randomImages[18];
        images7[2] = randomImages[19];
        images7[3] = randomImages[20];


        // set up the image arrays used as image buttons
        playArray[0] = getImage(getDocumentBase(), "file:///C:/play57.png");
        playArray[1] = getImage(getDocumentBase(), "file:///C:/play68.png");

        pauzeArray[0] = getImage(getDocumentBase(), "file:///C:/pauze57.png");
        pauzeArray[1] = getImage(getDocumentBase(), "file:///C:/pauze68.png");

        rewindArray[0] = getImage(getDocumentBase(), "file:///C:/rewind57.png");
        rewindArray[1] = getImage(getDocumentBase(), "file:///C:/rewind68.png");

        fastforwardArray[0] = getImage(getDocumentBase(),
"file:///C:/fastforward57.png");
        fastforwardArray[1] = getImage(getDocumentBase(),
"file:///C:/fastforward68.png");

        instantreplayArray[0] = getImage(getDocumentBase(),
"file:///C:/instantreplay57.png");
        instantreplayArray[1] = getImage(getDocumentBase(),
"file:///C:/instantreplay68.png");

        slideshowArray[0] = getImage(getDocumentBase(), "file:///C:/slideshow57.png");
        slideshowArray[1] = getImage(getDocumentBase(), "file:///C:/slideshow68.png");

        skipcommercialArray[0] = getImage(getDocumentBase(),
"file:///C:/skipcommercial57.png");
        skipcommercialArray[1] = getImage(getDocumentBase(),
"file:///C:/skipcommercial68.png");

        programArray[0] = getImage(getDocumentBase(), "file:///C:/program57.png");
        programArray[1] = getImage(getDocumentBase(), "file:///C:/program68.png");

        Vector devices=CaptureDeviceManager.getDeviceList(null);
        System.out.println(devices.size());

        cdi = (CaptureDeviceInfo) devices.elementAt(2);
        System.out.println(cdi.getName());
        setLayout(null);
```

```
        PlayButton = new Button("Play");
        RewindButton = new Button("Rewind");
        PauzeButton = new Button("Pauze");
        FastForwardButton = new Button("Fast Forward");
        InstantReplayButton = new Button("Instant Replay");
        Skip30Button = new Button("Skip Commercial");
        Mask30Button = new Button("Hide Commercial");
        InstantReplayButton.setBounds(1050,290,120, 30);
        InstantReplayButton.setBackground(Color.gray);
        Skip30Button.setBounds(1050, 50, 120, 30);
        Skip30Button.setBackground(Color.gray);
        Mask30Button.setBounds(1050, 90, 120, 30);
        Mask30Button.setBackground(Color.gray);
        RewindButton.setBounds(1050,130,90,30);
        RewindButton.setBackground(Color.gray);
        PlayButton.setBounds(1050,170,90,30);
        PlayButton.setBackground(Color.gray);
        PauzeButton.setBounds(1050,210,90,30);
        PauzeButton.setBackground(Color.gray);
        FastForwardButton.setBounds(1050, 250, 90, 30);
        FastForwardButton.setBackground(Color.gray);
        //add(RewindButton);
        //add(PlayButton);
        //add(PauzeButton);
        //add(FastForwardButton);
        //add(InstantReplayButton);
        //add(Skip30Button);
        //add(Mask30Button);
        //testImage = new Image("file:///C:/b1.jpg");
        //t = new Button("will this work", forlittlet);
        //t.setMnemonic(KeyEvent.VK_D);
        //t.setActionCommand("disable");
        //t.setBounds(541, 25, 90,30);
        //add(t);
        //t.addActionListener(this);
        //t.setToolTipText("shfkdlfjsdklfj");
        RewindButton.addActionListener(this);
        PlayButton.addActionListener(this);
        PauzeButton.addActionListener(this);
        FastForwardButton.addActionListener(this);
        InstantReplayButton.addActionListener(this);
        Skip30Button.addActionListener(this);
        Mask30Button.addActionListener(this);
        setSize(1275,700);   //was 518,387
        setBackground(Color.white);

        //mediaPlayer1.setMediaLocation(cdi.getName());
        System.out.println(cdi.getName());
        //mediaPlayer1.setMediaLocation(cdi);
        //mediaPlayer1.setMediaLocation(new java.lang.String("file:///C:/Heroes.mpg
        try {
            mediaPlayer1.setDataSource(Manager.createDataSource(new
MediaLocator("file:///" + playbackFile)));
        } catch(IOException e) {
            System.err.println("IOException");
```

```
        } catch(NoDataSourceException f) {
            System.err.println("NoDataSourceException");
        }

        add(mediaPlayer1);
        mediaPlayer1.setBounds(276,24,723,572); // was 36,24,423,272
        addMouseListener(this);
        rewindThread = new Thread(this);
        rewindThread.start();
        slideShowThread = new Thread(this);

        dim = getSize();
        offscreen = createImage(dim.width, dim.height);
        bufferGraphics = offscreen.getGraphics();

        modeSelection = new CheckboxGroup();
        liveMode = new Checkbox("Live Mode", modeSelection, false);
        playbackMode = new Checkbox("Playback Mode", modeSelection, true);
        liveMode.setBounds(397, 660, 120, 30);
        playbackMode.setBounds(517, 660, 120, 30);
        add(liveMode);
        add(playbackMode);
        fileName = new TextField(playbackFile, 240);
        fileName.setBounds(640, 665, 240, 30);
        add(fileName);
        commercialsGone = new Checkbox("Auto Remove Commercials", false);
        commercialsGone.setBounds(398, 680, 240, 30);
        add(commercialsGone);
        setupCommercialArrays(commercialDetectionFile);

    }


    /* Thie setupCommercialArrays fills an array with locations of
     *commercial blocks in an MPEG. It creates this data from a
     *text file generated by the Frame Grabber program
     */
    public void setupCommercialArrays(String commercialFile){
        //System.out.println("file:///" + commercialFile);
        for(int i = 0; i < 14; i++){
            commercialArray[i] = 0.0;
        }
        try { BufferedReader in = new BufferedReader(new FileReader( commercialFile));
        String stringReader;
        int spaceLocation;
        double time;
        double numberFrames;
        int commercialArrayIndex = 0;
        while((stringReader = in.readLine()) != null ){
            spaceLocation = stringReader.indexOf(" ");
            time = Double.parseDouble(stringReader.substring(0, spaceLocation));
            numberFrames = Double.parseDouble(stringReader.substring(spaceLocation+1,
stringReader.length()));
            //System.out.println(time);
            //System.out.println(numberFrames);
```

```
            if(numberFrames > 20){
                //System.out.println(time);
                commercialArray[commercialArrayIndex] = time;
                commercialArrayIndex++;
            }
        }
        in.close();
        } catch (IOException e) {}
    }

    /**
     *Get all the images in the locationDirectory that have a jpg extension
     */
    public void getImagesFromFile(java.lang.String locationDirectory, Image[] imageArray,
int imageArraySize) {
        File dir = new File(locationDirectory);
        FilenameFilter jpgFilter = new FilenameFilter() {
            public boolean accept(File dir, String name) {
                return name.endsWith(".jpg");
            }
        };

        String[] imagesInDirectory = dir.list(jpgFilter);
        if (imagesInDirectory != null){
            int i = 0;
            for(int j = 0; j < imageArraySize;j++){
                imageArray[j] = getImage(getDocumentBase(), "file:///" +
locationDirectory + "/" + imagesInDirectory[i]);
                if(i == imagesInDirectory.length-1){
                    i = 0;
                }
                else {
                    i++;
                }
            }
        }
        else{
            System.out.println("No images found for Slide Show");
        }


    }


    /* Set up the user interface */
    public void paint(Graphics g) {
        Image logo = getImage(getDocumentBase(),new
java.lang.String("file:///C:/logo.jpg"));
        Image logoFlipped = getImage(getDocumentBase(),new
java.lang.String("file:///C:/logoflipped.jpg"));
        Image logoFlipped2 = getImage(getDocumentBase(),new
java.lang.String("file:///C:/logo2.jpg"));
        Image logoFlipped3 = getImage(getDocumentBase(),new
java.lang.String("file:///C:/logo3.jpg"));
        bufferGraphics.drawImage(logo, 0, 0, this);
```

```
        bufferGraphics.drawImage(logoFlipped3, 1150, 0, this);
        bufferGraphics.drawImage(logoFlipped, 1150, 625, this);
        bufferGraphics.drawImage(logoFlipped2, 0, 625, this);
        //testImage = getImage(getDocumentBase(), new
java.lang.String("file:///C:/play57.png"));

        //Graphics q = testImage.getGraphics();
        //q.setColor(new Color(060, 060, 060));

        bufferGraphics.drawImage(rewindArray[rewindArrayIndex], 397, 580, 120, 40, this);
        bufferGraphics.drawImage(playArray[playArrayIndex], 517, 580, 120, 40, this);
        bufferGraphics.drawImage(pauzeArray[pauzeArrayIndex], 637, 580, 120, 40, this);
        bufferGraphics.drawImage(fastforwardArray[fastforwardArrayIndex], 757, 580, 120,
40, this);
        bufferGraphics.drawImage(instantreplayArray[instantreplayArrayIndex], 397, 620,
120, 40, this);
        bufferGraphics.drawImage(skipcommercialArray[skipcommercialArrayIndex], 517, 620,
120, 40, this);
        bufferGraphics.drawImage(slideshowArray[slideshowArrayIndex], 637, 620, 120, 40,
this);
        bufferGraphics.drawImage(programArray[programArrayIndex], 757, 620, 120, 40,
this);
//bufferGraphics.drawImage(images[imageIndex], 460, 80, 300, 200, this);  // was 460, 80
or 500, 90
        //bufferGraphics.drawImage(images4[imageIndex4], 590, 220, 300, 200, this); //
was590, 220 or 500, 380
        //bufferGraphics.drawImage(images2[imageIndex2], 190, 115, 300, 200, this); //
190, 115 or 310 230
        //bufferGraphics.drawImage(images3[imageIndex3], 825, 100, 300, 200, this); //
825, 100 or 645 230
        //bufferGraphics.drawImage(images5[imageIndex5], 370, 370, 300, 200, this);
        //bufferGraphics.drawImage(images6[imageIndex6], 790, 335, 300, 200, this);
//        //bufferGraphics.drawImage(images7[imageIndex7], 90, 300, 300, 200, this);
        bufferGraphics.drawImage(images7[imageIndex7], 90, 300, 300, 200, this);
        bufferGraphics.drawImage(images6[imageIndex6], 790, 335, 300, 200, this);
        bufferGraphics.drawImage(images5[imageIndex5], 460, 80, 300, 200, this);  // was
460, 80 or 500, 90
        bufferGraphics.drawImage(images4[imageIndex4], 190, 115, 300, 200, this); // 190,
115 or 310 230
        bufferGraphics.drawImage(images3[imageIndex3], 825, 100, 300, 200, this); // 825,
100 or 645 230
        bufferGraphics.drawImage(images2[imageIndex2], 590, 220, 300, 200, this); //
was590, 220 or 500, 380
        bufferGraphics.drawImage(images[imageIndex], 370, 370, 300, 200, this);




        g.drawImage(offscreen, 0, 0, this);
        //g.drawImage(fancyPlay, 356, 605, this);
    }

    public void update(Graphics g){
        paint(g);
    }
```

```
    public void stop() {
        //rewindThread.stop();
        rewindThread = null;
        if(mediaPlayer1 != null)
            mediaPlayer1.stop();
        else
            System.out.println("mediaPlayer1 is null");
    }

    public void destroy() {
        if(mediaPlayer1 != null) {
            mediaPlayer1.deallocate();
            running = false;
            rewindThread = null;
        }
        else
            System.out.println("mediaPlayer1 is null");
    }

    public void start() {
        if(mediaPlayer1 != null) {
            mediaPlayer1.start();
        }
        else
            System.out.println("mediaPlayer1 is null");
    }

    /* Thread used to take care of some of the mediaplayer controls */
    public void run() {

        while(running){
            System.out.println("bufferedAhead: " + bufferedAhead);
            System.out.println("bufferedBehind: " + bufferedBehind);
            // the following if handels a known commercial break between 488 and 670

            for(int i = 0; i < 14 && commercialsGoneBoolean; i=i+2){

                if(/*commercialsGone.getState() &&*/commercialsGoneBoolean &&
                    mediaPlayer1.getMediaTime().getSeconds() > (commercialArray[i] - 1.0)
&&
                    mediaPlayer1.getMediaTime().getSeconds() < commercialArray[i+1] ){
                    System.out.println("commercial");
                    mediaPlayer1.stop();
                    //double temp = mediaPlayer1.getMediaTime().getSeconds();
                    //temp = temp + 183;
                    mediaPlayer1.setMediaTime(new Time(commercialArray[i+1]));
                    mediaPlayer1.start();
                }
            }

            if(lastCommand == "Pauze"){
                // check to make sure that the buffered ahead does not become greater
than 29 minutes
```

```
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                if((bufferedAhead + timeWaited) > 1740.0 && playingMode == "live"){
//1740
                    // cannot pauze any longer, force player into play mode
                    bufferedAhead = bufferedAhead + timeWaited;


                    lastCommand = "Play";
                    rewindReq = false;
                    System.out.println("PLAYING");
                    mediaPlayer1.stop();
                    mediaPlayer1.setRate(1);
                    //System.out.println("ttttttt");
                    mediaPlayer1.start();
                    lastPlayCheck = mediaPlayer1.getMediaTime().getSeconds();
                    playArrayIndex = 1;
                    pauzeArrayIndex = 0;
                    repaint();

                }
            }

            if(lastCommand == "Fastforward"){
                // check to make sure that the buffered ahead does not become less than 0
minutes
                // and that the buffered behind does not become greater than 29 minutes
                double currentMediaTime = mediaPlayer1.getMediaTime().getSeconds();
                double totalFFTime = currentMediaTime - ffStartTime;
                if((bufferedAhead - totalFFTime) < 5.0 && playingMode == "live"){
                    // Need to force the mediaplater into play mode
                    bufferedAhead = bufferedAhead - totalFFTime;

                    lastCommand = "Play";
                    rewindReq = false;
                    System.out.println("PLAYING");
                    mediaPlayer1.stop();
                    mediaPlayer1.setRate(1);
                    //System.out.println("ttttttt");
                    mediaPlayer1.start();
                    lastPlayCheck = mediaPlayer1.getMediaTime().getSeconds();
                    playArrayIndex = 1;
                    fastforwardArrayIndex = 0;
                    repaint();


                }

            }
            if(lastCommand == "Play"){
                // update bufferedBehind, we have been playing
                bufferedBehind = bufferedBehind +
(mediaPlayer1.getMediaTime().getSeconds() - lastPlayCheck);
                lastPlayCheck = mediaPlayer1.getMediaTime().getSeconds();
            }
```

```
if(slideShowActive){
     if(slideShowCounter % 2 == 0 && slideShowCounter != 0){
         if(slideShowCounter == 2){
            //System.out.println("Doing a slideshow");
            //slideShowCounter = 0;
            picturesShown++;
            if(picturesShown != 4){
                imageIndex = picturesShown;
                repaint();
            }
          }
         if(slideShowCounter == 4){
             System.out.println("#2");
             if(picturesShown != 4){
                 imageIndex2 = picturesShown;
                 repaint();
             }
         }

         if(slideShowCounter == 6){
             System.out.println("#2");
             if(picturesShown != 4){
                 imageIndex3 = picturesShown;
                 repaint();
             }
         }
         if(slideShowCounter == 8){
             System.out.println("#2");
             //slideShowCounter = 0;
             if(picturesShown != 4){
                 imageIndex4 = picturesShown;
                 repaint();
             }
         }



         if(slideShowCounter == 10){
             //slideShowCounter = 0;
             System.out.println("#2");
             if(picturesShown != 4){
                 imageIndex5 = picturesShown;
                 repaint();
             }
         }


         if(slideShowCounter == 12){
             System.out.println("#2");
             if(picturesShown != 4){
                 imageIndex6 = picturesShown;
                 repaint();
             }
         }
```

```
            if(slideShowCounter == 14){
                System.out.println("#3");
                slideShowCounter = 0;
                if(picturesShown != 4){
                    imageIndex7 = picturesShown;
                    repaint();
                }

            }



            slideShowCounter++;

        }
        else{
            slideShowCounter++;
        }
        if(picturesShown == 4){
            slideShowActive = false;
            imageIndex = 0;
            imageIndex2 = 0;
            imageIndex3 = 0;
            imageIndex4 = 0;
            imageIndex5 = 0;
            imageIndex6 = 0;
            imageIndex7 = 0;
            slideshowArrayIndex = 0;
           // repaint();
            slideShowCounter = 2;
            picturesShown = 0;
            mediaPlayer1.setBounds(276,24,723,572);
            repaint();
        }
}
if(rewindReq) {
    System.out.println("need to fix rewinding()");
    rewinding();
}

 // OLD MASKING BEGIN
 if(endMask){
    System.out.println("Masking ended");
    endMask = false;
    //mediaPlayer1.start();
    mediaPlayer1.setMediaTime(new Time(endMaskTime));

}
if(startMask) {
    System.out.println("Masking started");

   if(mediaPlayer1.getMediaTime().getSeconds() > 10 ){
        startMask = false;
        System.out.println("Need to end mask");
```

```
                    //startMask = false;
                    mediaPlayer1.stop();
                    //mediaPlayer1.setMediaTime(new Time(endMaskTime));
                    mediaPlayer1.setMediaLocation(new
java.lang.String("file:///C:/bob.mpg"));
                    //int t = 0;
                    //while(t < 500)
                    //    t = t + 1;
                    //mediaPlayer1.setMediaTime(new Time(endMaskTime));
                    //mediaPlayer1.start();
                    System.out.println(endMaskTime);

                    mediaPlayer1.start();
                    endMask = true;
                    //mediaPlayer1.setMediaTime(new Time(endMaskTime));
                    //mediaPlayer1.start();
                }
            }
            // OLD MASKING END

            try {
                rewindThread.sleep(1000);
            }
            catch(InterruptedException e) {
                System.out.println(e);
            }


            // while(running){
            //
            //     try{
            //         slideShowThread.sleep(5000);
            //     } catch (InterruptedException e){}
            //


             }

    }

    javax.media.bean.playerbean.MediaPlayer mediaPlayer1 = new
javax.media.bean.playerbean.MediaPlayer();

    public void actionPerformed(ActionEvent evt) {
        if(evt.getSource() == RewindButton) {
            if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
            }
            //rewinding();
            lastCommand = "Rewind";
            rewindReq = true;
            //running = true;
            System.out.println("REWINDING");
```

```
        }
        else if(evt.getSource() == PauzeButton) {
            //running = false;
            lastCommand = "Pauze";
            pauzeStartTime = System.currentTimeMillis() / 1000.0;

            rewindReq = false;
            mediaPlayer1.stop();
            mediaPlayer1.setRate(1);
            System.out.println("PAUZING");
        }
        else if(evt.getSource() == PlayButton) {
            //running = true;
            //System.out.println("lastCommand " + lastCommand);
            lastPlayCheck = mediaPlayer1.getMediaTime().getSeconds();
            if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
            }
            if(lastCommand == "Fastforward"){
                double timeFastforwarded = mediaPlayer1.getMediaTime().getSeconds() -
ffStartTime;
                //System.out.println("fastforwarded: " + timeFastforwarded);
                bufferedAhead = bufferedAhead - timeFastforwarded;
                bufferedBehind = bufferedBehind + timeFastforwarded;
            }
            lastCommand = "Play";
            try {
                rewindThread.sleep(500);
            }
            catch (InterruptedException e) {
                System.out.println(e);
            }
            rewindReq = false;
            System.out.println("PLAYING");
            mediaPlayer1.stop();
            mediaPlayer1.setRate(1);
            //System.out.println("ttttttt");
            mediaPlayer1.start();
            //System.out.println("ssssss");
        }
        else if(evt.getSource() == FastForwardButton) {
            //running = false;
            if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
            }
            if(bufferedAhead > 3.0){
                lastCommand = "Fastforward";
                ffStartTime = mediaPlayer1.getMediaTime().getSeconds();
                rewindReq = false;
                System.out.println("FASTFORWARDING");
                mediaPlayer1.stop();
```

```
                mediaPlayer1.setRate(2);
                mediaPlayer1.start();
            }
        }
        else if(evt.getSource() == InstantReplayButton) {
            rewindReq = false;
            if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
            }
            lastCommand = "Play";
            if(bufferedBehind >= 7.0){
                // Check to make sure there is at least a 5 second buffer
                bufferedBehind = bufferedBehind - 7.0;
                bufferedAhead = bufferedAhead + 7.0;
                System.out.println("INSTANT REPLAY");
                mediaPlayer1.stop();
                double temp;
                //System.out.println(temp);
                temp = mediaPlayer1.getMediaTime().getSeconds();
                //System.out.println(temp);
                temp = temp - 7;
                //System.out.println(mediaPlayer1.getTargetState());
                //System.out.println(mediaPlayer1.Started);
                if(mediaPlayer1.getTargetState() < mediaPlayer1.Started)
                    mediaPlayer1.prefetch();
                mediaPlayer1.setMediaTime(new Time(temp));
                mediaPlayer1.start();
            }
        }
        else if(evt.getSource() == Skip30Button) {
            rewindReq = false;
            if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
            }
            lastCommand = "Play";
            if(bufferedAhead >= 30){
                bufferedAhead = bufferedAhead - 30;
                bufferedBehind = bufferedBehind + 30;
                System.out.println("SKIP 30");
                mediaPlayer1.stop();
                double temp;
                temp = mediaPlayer1.getMediaTime().getSeconds();
                temp = temp + 30;
                if(mediaPlayer1.getTargetState() < mediaPlayer1.Started)
                    mediaPlayer1.prefetch();
                mediaPlayer1.setMediaTime(new Time(temp));
                mediaPlayer1.start();
            }
        }
        else if(evt.getSource()== Mask30Button) {
            lastCommand = "Play";
```

```
            System.out.println("MASK 30");
            // goto 30 second slide show
            //mediaPlayer1.setBounds(276,24,723,572);
            mediaPlayer1.setBounds(883,570,190,120);
            slideShow();
        }
    }


    /* start a slideshow by setting slieShowActive true, the thread will take care of the
rest */
    public void slideShow() {
        System.out.println("implement the slideshow here");
        //Q.drawImage(images[0], 30, 30, this);
        slideShowActive = true;

    }

    public void rewinding() {
            int temp = 0;
            if(rewindReq == false)
                return;
            mediaPlayer1.stop();
            double beginMediaTime = mediaPlayer1.getMediaTime().getSeconds();
            while(rewindReq){
            //mediaPlayer1.stop();
            //Control noth;
            //Control[] ccc = mediaPlayer1.getControls();
            //long dura = mediaPlayer1.getDuration().getNanoseconds();
                if(rewindNumber == 1){
                    //mediaPlayer1.stop();
                    double dura = mediaPlayer1.getMediaTime().getSeconds();
                    double newTime = dura - 0.1;
                    if(newTime > 0.0) {
                    mediaPlayer1.setMediaTime(new Time(newTime));
                    }
                    else {
                        rewindReq = false;
                    }
                    if(bufferedAhead + beginMediaTime -
mediaPlayer1.getMediaTime().getSeconds() > 1740)
                    {
                        // falling behind too far
                        // force play mode
                        rewindReq = false;
                    }
                    //if(mediaPlayer1.getTargetState() < mediaPlayer1.Started)
                    //    mediaPlayer1.prefetch();
                    rewindNumber++;
                    }
                else{
                    //mediaPlayer1.start();
                    rewindNumber++;
```

```
                if(rewindNumber == 1000000)
                    rewindNumber = -1000000;
            }
        //mediaPlayer1.start();
        }
        double change = beginMediaTime - mediaPlayer1.getMediaTime().getSeconds();
        bufferedAhead = bufferedAhead + change;
        bufferedBehind = bufferedBehind - change;
        mediaPlayer1.start();
        //mediaPlayer1.start();

    }


    /* When a mouse has been clicked, see what button was clicked and execute
     * the wanted routine
     */
    public void mouseClicked(MouseEvent me) {
        //System.out.println("mouseClicked");
        xpos = me.getX();
        ypos = me.getY();



        if(ypos >580 && ypos < 620){
            if(xpos > 397 && xpos < 517){ //rewind
                playArrayIndex = 0;
                fastforwardArrayIndex = 0;
                rewindArrayIndex = 1;
                pauzeArrayIndex = 0;
                instantreplayArrayIndex = 0;
                slideshowArrayIndex = 0;
                skipcommercialArrayIndex = 0;
                programArrayIndex = 0;
                repaint();

             if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
             }
             //rewinding();
             lastCommand = "Rewind";
             rewindReq = true;
             //running = true;
             System.out.println("REWINDING");

            }
            if(xpos > 517 && xpos < 637){ // play
                playArrayIndex = 1;
                fastforwardArrayIndex = 0;
                rewindArrayIndex = 0;
                pauzeArrayIndex = 0;
                instantreplayArrayIndex = 0;
                slideshowArrayIndex = 0;
```

```
                skipcommercialArrayIndex = 0;
                programArrayIndex = 0;
                repaint();

                lastPlayCheck = mediaPlayer1.getMediaTime().getSeconds();
                if(lastCommand == "Pauze"){
                    double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                    bufferedAhead = bufferedAhead + timeWaited;
                }
                if(lastCommand == "Fastforward"){
                    double timeFastforwarded = mediaPlayer1.getMediaTime().getSeconds() -
ffStartTime;
                    //System.out.println("fastforwarded: " + timeFastforwarded);
                    bufferedAhead = bufferedAhead - timeFastforwarded;
                    bufferedBehind = bufferedBehind + timeFastforwarded;
                }
                lastCommand = "Play";
                try {
                    rewindThread.sleep(500);
                }
                catch (InterruptedException e) {
                    System.out.println(e);
                }
                rewindReq = false;
                System.out.println("PLAYING");
                mediaPlayer1.stop();
                mediaPlayer1.setRate(1);
                //System.out.println("ttttttt");
                mediaPlayer1.start();
                //System.out.println("ssssss");
            }
            if(xpos > 637 && xpos < 757){ // pauze
                playArrayIndex = 0;
                fastforwardArrayIndex = 0;
                rewindArrayIndex = 0;
                pauzeArrayIndex = 1;
                instantreplayArrayIndex = 0;
                slideshowArrayIndex = 0;
                skipcommercialArrayIndex = 0;
                programArrayIndex = 0;
                repaint();

                lastCommand = "Pauze";
                pauzeStartTime = System.currentTimeMillis() / 1000.0;

                rewindReq = false;
                mediaPlayer1.stop();
                mediaPlayer1.setRate(1);
                System.out.println("PAUZING");
                //mediaPlayer1.stop();
                //mediaPlayer1.setMediaLocation(new
java.lang.String("file:///C:/Heroes.mpg"));
                //mediaPlayer1.start();
```

```
            }
            if(xpos > 757 && xpos < 877){ // ff


                if(lastCommand == "Pauze"){
                    double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                    bufferedAhead = bufferedAhead + timeWaited;
                }
                if(bufferedAhead > 3.0 || playingMode == "playback"){
                    playArrayIndex = 0;
                    fastforwardArrayIndex = 1;
                    rewindArrayIndex = 0;
                    pauzeArrayIndex = 0;
                    instantreplayArrayIndex = 0;
                    slideshowArrayIndex = 0;
                    skipcommercialArrayIndex = 0;
                    programArrayIndex = 0;
                    repaint();

                    lastCommand = "Fastforward";
                    ffStartTime = mediaPlayer1.getMediaTime().getSeconds();
                    rewindReq = false;
                    System.out.println("FASTFORWARDING");
                    mediaPlayer1.stop();
                    mediaPlayer1.setRate(2);
                    mediaPlayer1.start();
                }



            }
        }
        if(ypos > 620 && ypos < 660){
            if(xpos > 397 && xpos < 517){ // inst replay
                playArrayIndex = 0;
                fastforwardArrayIndex = 0;
                rewindArrayIndex = 0;
                pauzeArrayIndex = 0;
                instantreplayArrayIndex = 1;
                slideshowArrayIndex = 0;
                skipcommercialArrayIndex = 0;
                programArrayIndex = 0;
                repaint();

                rewindReq = false;
                if(lastCommand == "Pauze"){
                    double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                    bufferedAhead = bufferedAhead + timeWaited;
                }
                lastCommand = "Play";
                if(bufferedBehind >= 7.0 || playingMode == "playback"){
                    // Check to make sure there is at least a 5 second buffer
                    bufferedBehind = bufferedBehind - 7.0;
```

```
                bufferedAhead = bufferedAhead + 7.0;
                System.out.println("INSTANT REPLAY");
                mediaPlayer1.stop();
                double temp;
                //System.out.println(temp);
                temp = mediaPlayer1.getMediaTime().getSeconds();
                //System.out.println(temp);
                temp = temp - 7;
                //System.out.println(mediaPlayer1.getTargetState());
                //System.out.println(mediaPlayer1.Started);
                if(mediaPlayer1.getTargetState() < mediaPlayer1.Started)
                    mediaPlayer1.prefetch();
                mediaPlayer1.setMediaTime(new Time(temp));
                mediaPlayer1.start();
            }
            playArrayIndex = 1;
            instantreplayArrayIndex = 0;
            repaint();

        }
        if(xpos > 517 && xpos < 637){ // skip
            playArrayIndex = 0;
            fastforwardArrayIndex = 0;
            rewindArrayIndex = 0;
            pauzeArrayIndex = 0;
            instantreplayArrayIndex = 0;
            slideshowArrayIndex = 0;
            skipcommercialArrayIndex = 1;
            programArrayIndex = 0;
            repaint();

            rewindReq = false;
            if(lastCommand == "Pauze"){
                double timeWaited = (System.currentTimeMillis() / 1000.0) -
pauzeStartTime;
                bufferedAhead = bufferedAhead + timeWaited;
            }
            lastCommand = "Play";
            if(bufferedAhead >= 30 || playingMode == "playback"){
                bufferedAhead = bufferedAhead - 30;
                bufferedBehind = bufferedBehind + 30;
                System.out.println("SKIP 30");
                mediaPlayer1.stop();
                double temp;
                temp = mediaPlayer1.getMediaTime().getSeconds();
                temp = temp + 30;
                if(mediaPlayer1.getTargetState() < mediaPlayer1.Started)
                    mediaPlayer1.prefetch();
                mediaPlayer1.setMediaTime(new Time(temp));
                mediaPlayer1.start();
            }

        }
        if(xpos > 637 && xpos < 757){ // slideshow
```

```
    slideshowArrayIndex = 1;

    repaint();

    lastCommand = "Play";


    System.out.println("MASK 30");
    // goto 30 second slide show
    //mediaPlayer1.setBounds(276,24,723,572);
    mediaPlayer1.setBounds(883,570,190,120);
    slideShow();

}
if(xpos > 757 && xpos < 877){ // program
    playArrayIndex = 1;
    fastforwardArrayIndex = 0;
    rewindArrayIndex = 0;
    pauzeArrayIndex = 0;
    instantreplayArrayIndex = 0;
    slideshowArrayIndex = 0;
    skipcommercialArrayIndex = 0;
    programArrayIndex = 0;
    repaint();
    if(liveMode.getState()){
            // live mode requested
        System.out.println("livelivelivelivelivelive");
        mediaPlayer1.stop();
        mediaPlayer1.deallocate();
        int waitSome = 100000;
        while(waitSome > 0)
            waitSome = waitSome - 1;
        mediaPlayer1.setMediaLocation("file:///C:/live.mpg");
        //if(mediaPlayer1.getTargetState() < mediaPlayer1.Started)
        //    mediaPlayer1.prefetch();
        int waiter = 100000;
        while(waiter > 0)
            waiter = waiter - 1;
        mediaPlayer1.start();
        repaint();
    }
    else{
            // playback mode requested
        System.out.println("playbackplaybackplayback");
        mediaPlayer1.stop();
        int waitSome = 100000;
        while(waitSome > 0)
            waitSome = waitSome - 1;
        //if(fileName.getText() != lastFile)
        //{
            mediaPlayer1.setMediaLocation("file:///" + fileName.getText());
            int waiter = 100000;
            while(waiter > 0)
                waiter = waiter - 1;
            mediaPlayer1.start();
```

```
                    lastFile = fileName.getText();
                //}
            }
            if(commercialsGone.getState() == true)
            {
                commercialsGoneBoolean = true;
            }
            else
            {
                commercialsGoneBoolean = false;
            }
        }
    }
}

public void mouseEntered(MouseEvent me) {
    System.out.println("mouseEntered");
    if(slideShowActive){
        slideShowActive = false;
        imageIndex = 0;
        imageIndex2 = 0;
        imageIndex3 = 0;
        imageIndex4 = 0;
        imageIndex5 = 0;
        imageIndex6 = 0;
        imageIndex7 = 0;
        //playArrayIndex = 1;
        slideshowArrayIndex = 0;
        repaint();
        slideShowCounter = 2;
        picturesShown = 0;
        mediaPlayer1.setBounds(276,24,723,572);
    }

}

public void mousePressed(MouseEvent me) {
    //System.out.println("mousePressed");
    //popper.show(me.getComponent(), me.getX(), me.getY());
}

public void mouseReleased(MouseEvent me) {
    //System.out.println("mouseReleased");
}

public void mouseExited(MouseEvent me) {
   // System.out.println("mouseexited");
   // System.out.println(me.getX());
}


}
```

FRAMEGRABBER.JAVA

```
/*
 * FrameAccess.java
 *
 *
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

/**
 *
 * @author Irene T
 */

import java.awt.*;
import javax.media.*;
import javax.media.control.TrackControl;
import javax.media.Format;
import javax.media.format.*;
import java.io.*;
import javax.imageio.*;
import javax.imageio.stream.*;
import java.awt.image.*;
import java.util.*;
import javax.media.util.*;

/**
 *Program which accesses every frame in an MPEG by using a pass-
 *thru codec. This program creates a textfile which lists
 *groups of black frames in an MPEG. It list the location
 *in the MPEG and the number of frames per group
*/
public class FrameAccess implements ControllerListener {

   Processor p;
   Object waitSync = new Object();
   boolean stateTransitionOK = true;
   public boolean alreadyPrnt = false;
        boolean allBlack = false;
        static BufferedWriter out;
        int blackCount = 0;
        boolean lastBlack = false;

   /**
   * Given a media locator, create a processor and use that processor
   * as a player to playback the media.
   *
   * During the processor's Configured state, two "pass-thru" codecs,
   * PreAccessCodec and PostAccessCodec, are set on the video track.
   * These codecs are used to get access to individual video frames
   * of the media.
   *
   *
```

```
*/
public boolean open(MediaLocator ml) {

   try {
      p = Manager.createProcessor(ml);
   } catch (Exception e) {
      System.err.println(
         "Failed to create a processor from the given url: " + e);
      return false;
   }

   p.addControllerListener(this);

   // Put the Processor into configured state.
   p.configure();
   if (!waitForState(Processor.Configured)) {
      System.err.println("Failed to configure the processor.");
      return false;
   }

   // So I can use it as a player.
   p.setContentDescriptor(null);

   // Obtain the track controls.
   TrackControl tc[] = p.getTrackControls();

   if (tc == null) {
      System.err.println(
         "Failed to obtain track controls from the processor.");
      return false;
   }

   // Search for the track control for the video track.
   TrackControl videoTrack = null;

   for (int i = 0; i < tc.length; i++) {
      if (tc[i].getFormat() instanceof VideoFormat) videoTrack = tc[i];
      else  tc[i].setEnabled(false);
   }

   if (videoTrack == null) {
      System.err.println("The input media does not contain a video track.");
      return false;
   }
   String videoFormat = videoTrack.getFormat().toString();
   Dimension videoSize = parseVideoSize(videoFormat);
   System.err.println("Video format: " + videoFormat);

   // Instantiate and set the frame access codec to the data flow path.
   try {
      Codec codec[] = { new PostAccessCodec(videoSize)};
      videoTrack.setCodecChain(codec);
   } catch (UnsupportedPlugInException e) {
      System.err.println("The process does not support effects.");
   }
```

```
      // Realize the processor.
      p.prefetch();
      if (!waitForState(Processor.Prefetched)) {
         System.err.println("Failed to realise the processor.");
         return false;
      }

      p.start();
      return true;
   }

   /**parse the size of the video from the string videoformat*/
   public Dimension parseVideoSize(String videoSize){
      int x=300, y=200;
      StringTokenizer strtok = new StringTokenizer(videoSize, ", ");
      strtok.nextToken();
      String size = strtok.nextToken();
      StringTokenizer sizeStrtok = new StringTokenizer(size, "x");
      try{
         x = Integer.parseInt(sizeStrtok.nextToken());
         y = Integer.parseInt(sizeStrtok.nextToken());
      } catch (NumberFormatException e){
         System.out.println("unable to find video size, assuming default of 300x200");
      }
      System.out.println("Image width  = " + String.valueOf(x) +"\nImage height = "+
String.valueOf(y));
      return new Dimension(x, y);
   }


   /**
   * Block until the processor has transitioned to the given state.
   * Return false if the transition failed.
   */
   boolean waitForState(int state) {
      synchronized (waitSync) {
         try {
            while (p.getState() != state && stateTransitionOK)
               waitSync.wait();
         } catch (Exception e) {
         }
      }
      return stateTransitionOK;
   }

   /**
   * Controller Listener.
   */
   public void controllerUpdate(ControllerEvent evt) {

      if (evt instanceof ConfigureCompleteEvent
         || evt instanceof RealizeCompleteEvent
         || evt instanceof PrefetchCompleteEvent) {
         synchronized (waitSync) {
```

```
            stateTransitionOK = true;
            waitSync.notifyAll();
         }
      } else if (evt instanceof ResourceUnavailableEvent) {
         synchronized (waitSync) {
            stateTransitionOK = false;
            waitSync.notifyAll();
         }
      } else if (evt instanceof EndOfMediaEvent) {
         p.close();
         System.exit(0);
      }
   }

   /**
   * Main program
   */
   public static void main(String[] args) {

      //if (args.length == 0) {
                //    System.err.println("I still dont know what iam doing");
      // prUsage();
   //    System.exit(0);
   // }

   // String url = args[0];


           try {
                out = new BufferedWriter(new FileWriter("Southpark.txt"));
                out.write("");
                out.close();
                } catch (IOException e) {

                }


                String url = "file:///C:/Heroes.mpg";

      if (url.indexOf(":") < 0) {
                        System.err.println("need :");
         prUsage();
         System.exit(0);
      }

      MediaLocator ml;

      if ((ml = new MediaLocator(url)) == null) {
         System.err.println("Cannot build media locator from: " + url);
         System.exit(0);
      }

      FrameAccess fa = new FrameAccess();
```

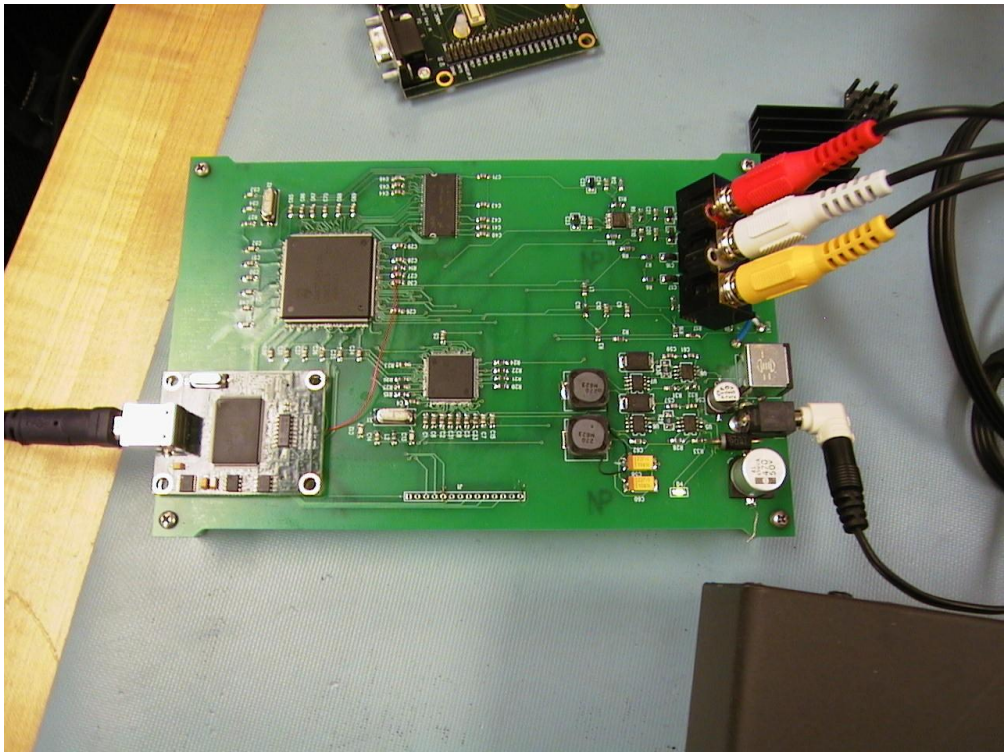
```
   if (!fa.open(ml))
      System.exit(0);
}

static void prUsage() {
   System.err.println("Usage: java FrameAccess <url>");
}

/**************************************************
* Inner class.
*
* A pass-through codec to access to individual frames.
**************************************************/

public class PreAccessCodec implements Codec {

   /**
   * Callback to access individual video frames.
   */
   void accessFrame(Buffer frame) {

      // For demo, we'll just print out the frame #, time &
      // data length.

      long t = (long) (frame.getTimeStamp() / 10000000f);

      System.err.println(
         "Pre: frame #: "
            + frame.getSequenceNumber()
            + ", time: "
            + ((float) t) / 100f
            + ", len: "
            + frame.getLength());
   }

   /**
   * The code for a pass through codec.
   */

   // We'll advertize as supporting all video formats.
   protected Format supportedIns[] = new Format[] { new VideoFormat(null)};

   // We'll advertize as supporting all video formats.
   protected Format supportedOuts[] = new Format[] { new VideoFormat(null)};

   Format input = null, output = null;

   public String getName() {
      return "Pre-Access Codec";
   }

   //these dont do anything
   public void open() {}
   public void close() {}
```

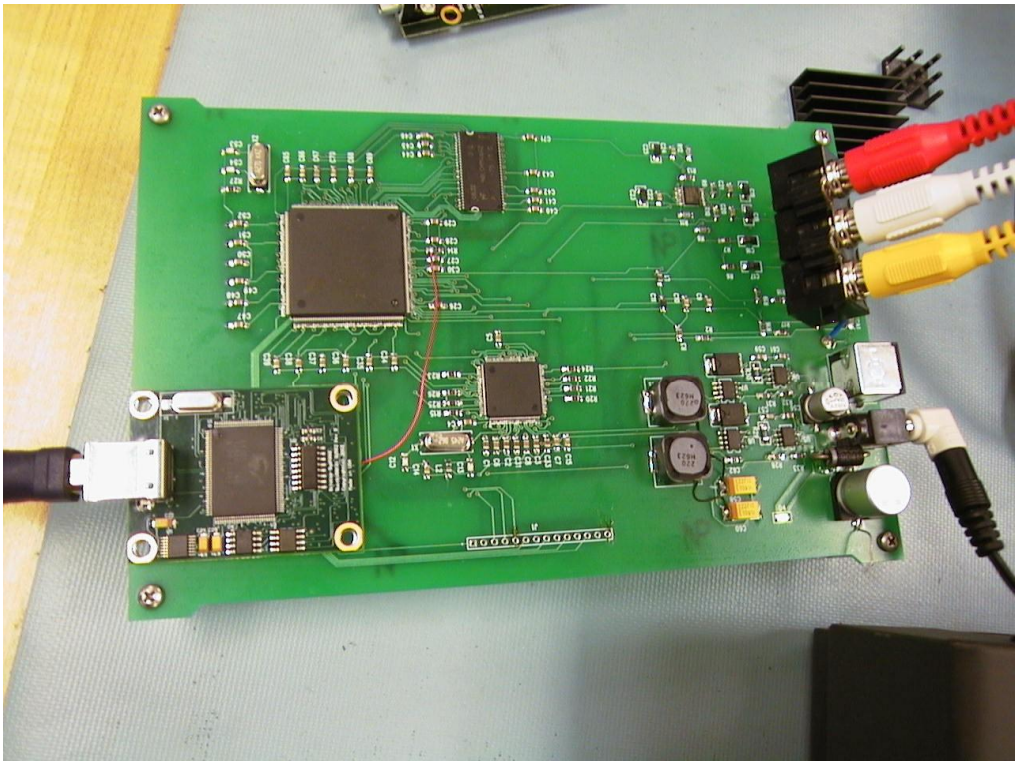
```
public void reset() {}

public Format[] getSupportedInputFormats() {
   return supportedIns;
}

public Format[] getSupportedOutputFormats(Format in) {
   if (in == null)
      return supportedOuts;
   else {
      // If an input format is given, we use that input format
      // as the output since we are not modifying the bit stream
      // at all.
      Format outs[] = new Format[1];
      outs[0] = in;
      return outs;
   }
}

public Format setInputFormat(Format format) {
   input = format;
   return input;
}

public Format setOutputFormat(Format format) {
   output = format;
   return output;
}

public int process(Buffer in, Buffer out) {

   // This is the "Callback" to access individual frames.
   accessFrame(in);

   // Swap the data between the input & output.
   Object data = in.getData();
   in.setData(out.getData());
   out.setData(data);

   // Copy the input attributes to the output
   out.setFlags(Buffer.FLAG_NO_SYNC);
   out.setFormat(in.getFormat());
   out.setLength(in.getLength());
   out.setOffset(in.getOffset());

   return BUFFER_PROCESSED_OK;
}

public Object[] getControls() {
   return new Object[0];
}

public Object getControl(String type) {
   return null;
}
```

```
   }

   public class PostAccessCodec extends PreAccessCodec {
      // We'll advertize as supporting all video formats.
      public PostAccessCodec(Dimension size) {
         supportedIns = new Format[] { new RGBFormat()};
         this.size = size;
      }

      /**
      * Callback to access individual video frames.
      */
      void accessFrame(Buffer frame) {

         // For demo, we'll just print out the frame #, time &
         // data length.
         if (!alreadyPrnt) {
            BufferToImage stopBuffer = new BufferToImage((VideoFormat)
frame.getFormat());
            Image stopImage = stopBuffer.createImage(frame);
            try {
               BufferedImage outImage = new BufferedImage(size.width, size.height,
BufferedImage.TYPE_INT_RGB);
               Graphics og = outImage.getGraphics();
               og.drawImage(stopImage, 0, 0, size.width, size.height, null);
               //prepareImage(outImage,rheight,rheight, null);
               Iterator writers = ImageIO.getImageWritersByFormatName("jpg");
               ImageWriter writer = (ImageWriter) writers.next();

               //Once an ImageWriter has been obtained, its destination must be set to an
ImageOutputStream:

                                          File f = new File("holder.jpg");

                                          //File f = new File(frame.getSequenceNumber() +
".jpg");
               ImageOutputStream ios = ImageIO.createImageOutputStream(f);
               writer.setOutput(ios);

               //Finally, the image may be written to the output stream:
               //BufferedImage bi;
               //writer.write(imagebi);
               writer.write(outImage);
               ios.close();

                                          BufferedImage img = null;
                                          img = ImageIO.read(new File("holder.jpg"));
                                          int[] pixies = new int[100000];
                                          allBlack = false;
                                          // black pixel = -16777216
                                          img.getRGB(150, 170, 10, 10, pixies, 0, 10 );
                                          int blah = img.getRGB(300, 230);
                                          int x = 0;
                                          int y = 0;
                                          boolean all_black = true;
```

```
                                        boolean done = false;
                                        while(all_black && !done) {
                                            if(img.getRGB(x,y) != -16777216) {
                                                all_black = false;
                                            }
                                            x = x + 1;
                                            if(x == 300){
                                                 x = 0;
                                                 y = y + 1;
                                            }
                                            if(y == 230){
                                                 done = true;
                                            }

                                        }
                                        if(all_black) {
                                               lastBlack = true;
                                               blackCount++;
                                        }
                                        else {
                                            //We only want to write to the file if we
have
                                            // seen the end of a group of  black frames
                                            if(lastBlack){
                                                long t = (long) (frame.getTimeStamp() /
10000000f);
                                                //System.out.println("writing to file");
                                                //System.err.println(
                                                try{
                                                    out = new BufferedWriter(new
FileWriter("Heroes.txt", true));

                                                    out.write( "" + ((float) t) / 100f +
" " + blackCount + "\n");
                                                    blackCount = 0;
                                                    lastBlack = false;
                                                   // "Post: frame #: "
                                                   // + frame.getSequenceNumber()
                                                   // + ", time: "
                                                   // + ((float) t) / 100f
                                                   // + ", len: "
                                                   // + frame.getLength()+ "\n");
                                                    out.close();


                                            } catch (IOException e) {}
                                            }
                                        }



                    } catch (IOException e) {
                       System.out.println("Error :" + e);
                    }
                }
```

```
         //alreadyPrnt = true;
         long t = (long) (frame.getTimeStamp() / 10000000f);
                        if(allBlack == true) {
         System.err.println(
            "Post: frame #: "
               + frame.getSequenceNumber()
               + ", time: "
               + ((float) t) / 100f
               + ", len: "
               + frame.getLength());
                        }
      }

      public String getName() {
         return "Post-Access Codec";
      }
      private Dimension size;
   }
}



    /** Creates a new instance of FrameAccess */
  //  public FrameAccess() {
  //  }
```





APPENDIX F – HARDWARE SCHEMATICS

The following eight pages contain schematics for:

DVR SCHEMATIC

PCB DESIGN LAYOUT

PCB TOP LAYER

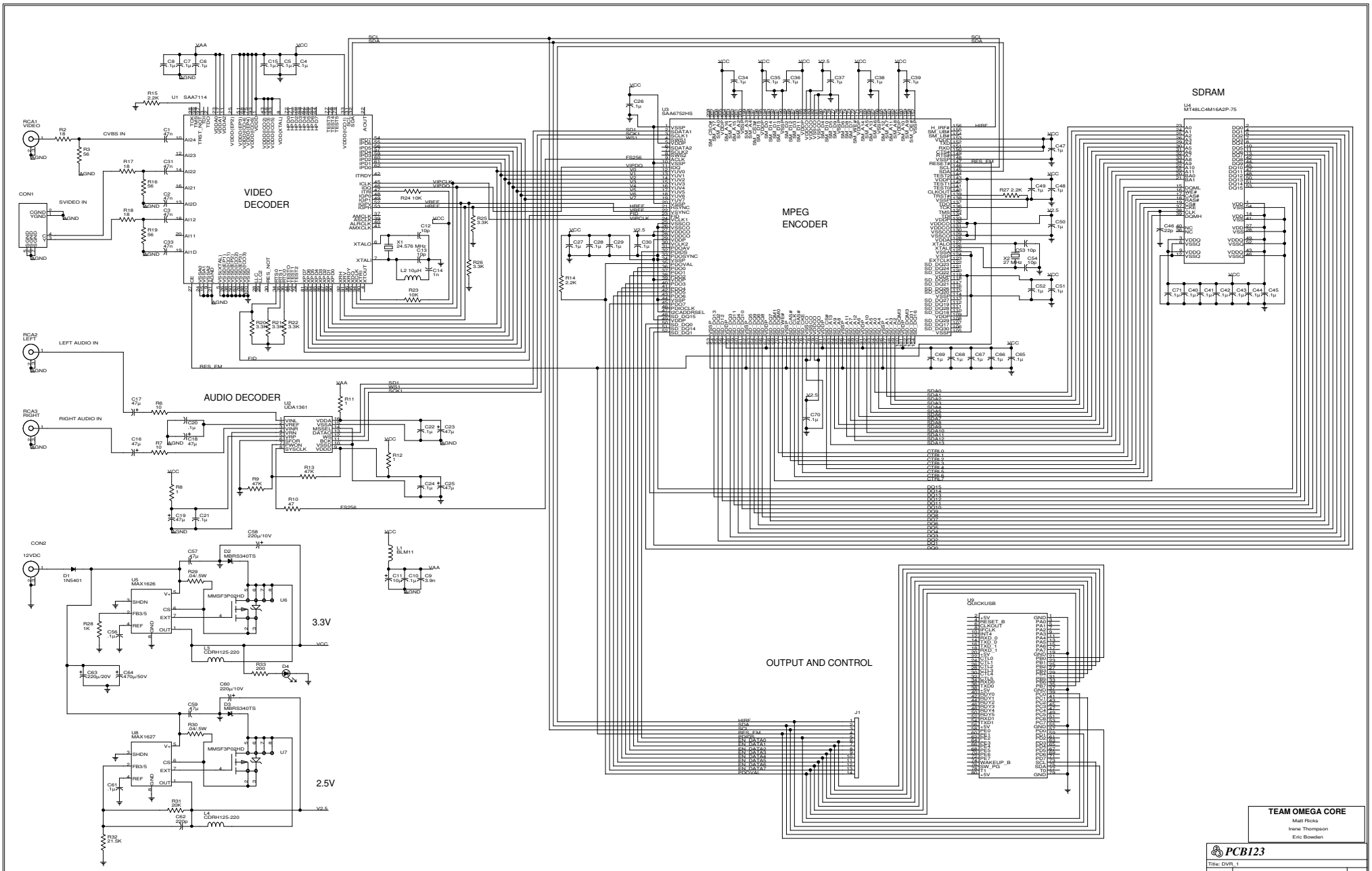
PCB BOTTOM LAYER

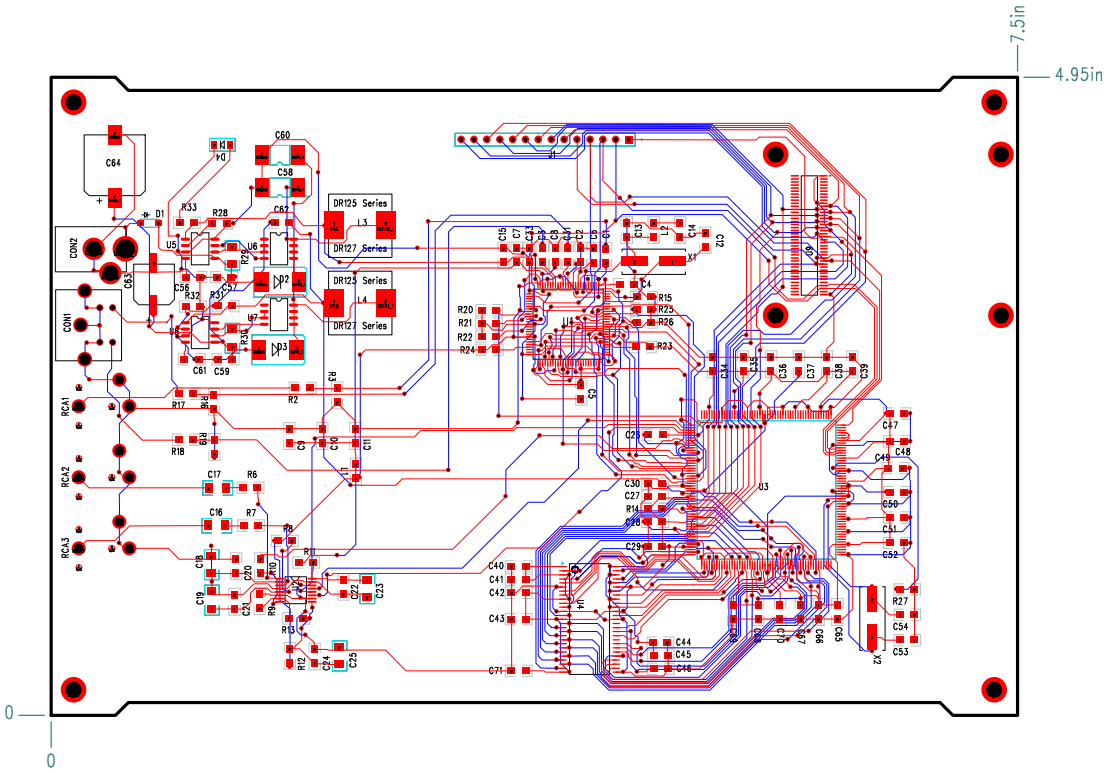
PCB TOP LAYER SOLDER MASK

PCB BOTTOM LAYER SOLDER MASK

PCB DRILL POINTS

PCB SILK SCREEN


VIDEO DECODER
MPEG ENCODER
SDRAM
AUDIO DECODER
OUTPUT AND CONTROL
U1 SAA7114
U3 SAA6752HS
U4 MT48LC4M16A2P-75
U2 UDA1361
U9 QUICKUSB
U5 MAX1626
U8 MAX1627
U6
U7
CON1
CON2
RCA1 VIDEO
RCA2 LEFT
RCA3 RIGHT
CVBS IN
SVIDEO IN
LEFT AUDIO IN
RIGHT AUDIO IN
12VDC
D1 1N5401
D2 MBRS340TS
D3 MBRS340TS
3.3V
2.5V
J1
RES_EM
FID
FS256
TEAM OMEGA CORE
Matt Ricks
Irene Thompson
Eric Bowden
PCB123
Title: DVR_1
Size D
Document Number
Rev A
Date: Fri, December 08, 2006



Drill Chart				
Sym	Size	Qty	Tol	Type
+	0.02in	457	+/-0.004"	Plated
×	0.033in	14	+/-0.004"	Plated
⊞	0.04in	4	+/-0.004"	Plated
A	0.0492in	6	+/-0.004"	Non-Plated
B	0.052in	3	+/-0.004"	Non-Plated
⧖	0.079in	9	+/-0.004"	Plated
△	0.093in	3	+/-0.004"	Plated
◈	0.125in	10	+/-0.004"	Plated
⋌	0.15in	1	+/-0.004"	Plated

Board Stackup

0.062" Overall Thickness

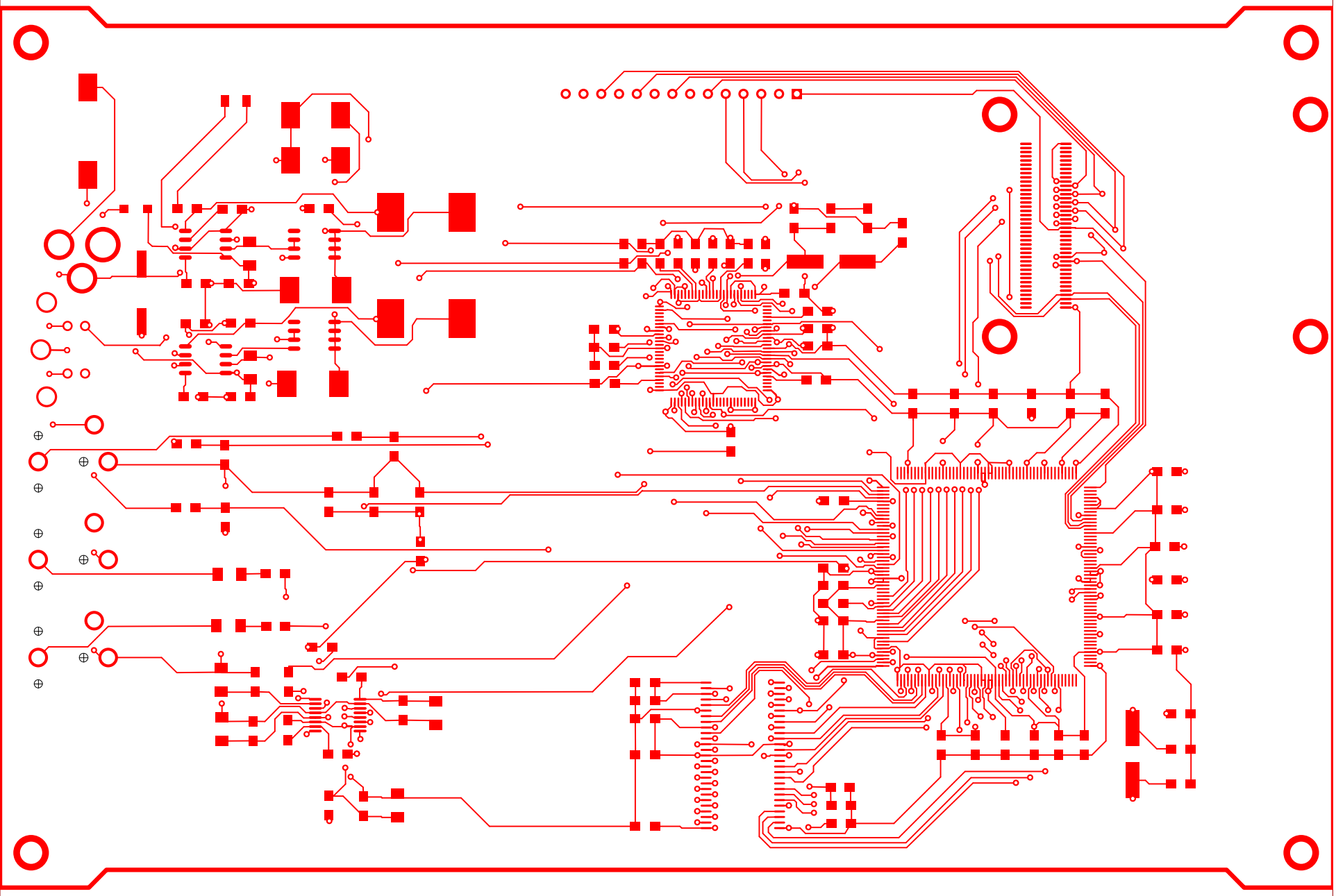
Untitled

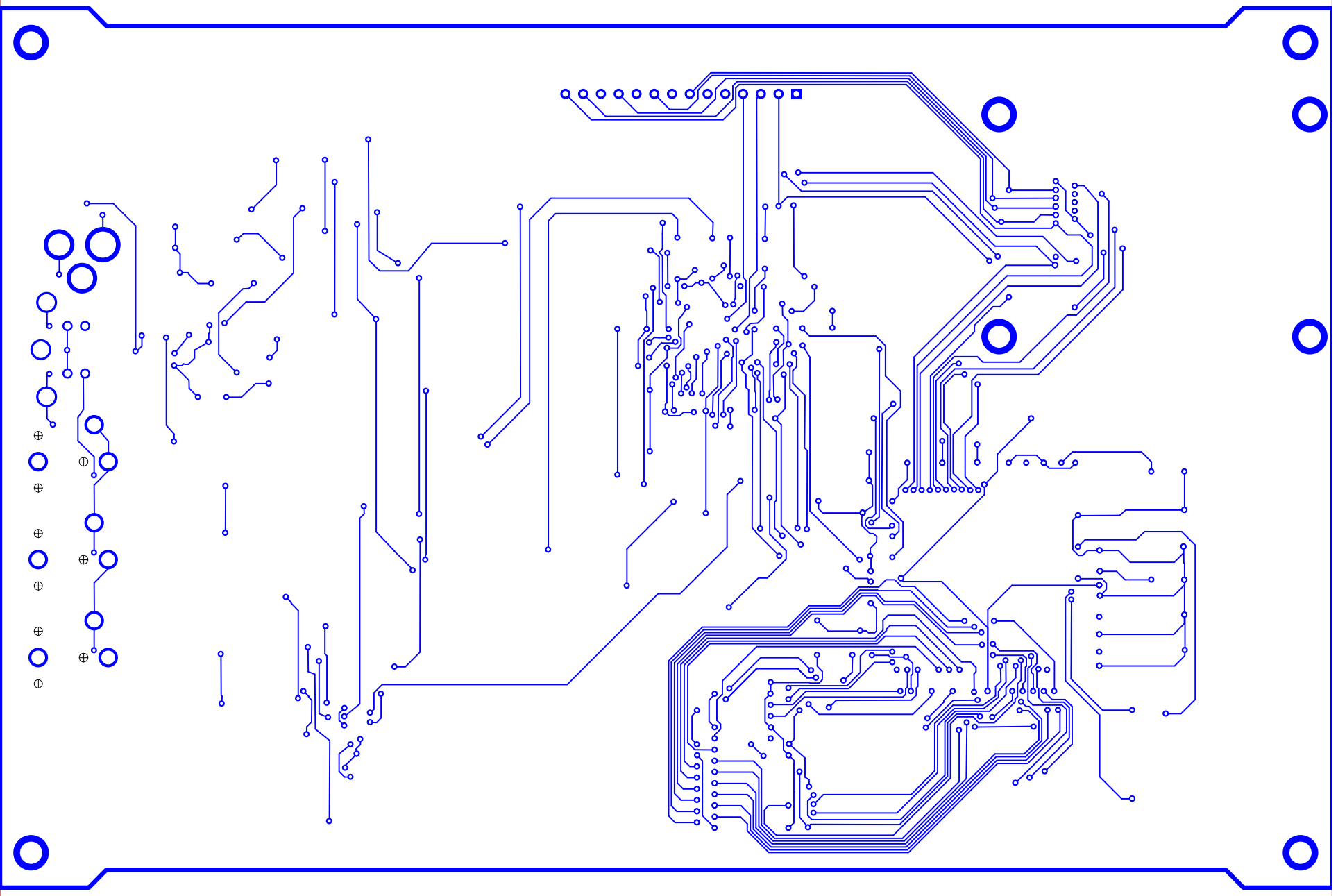
Rev: 1.0

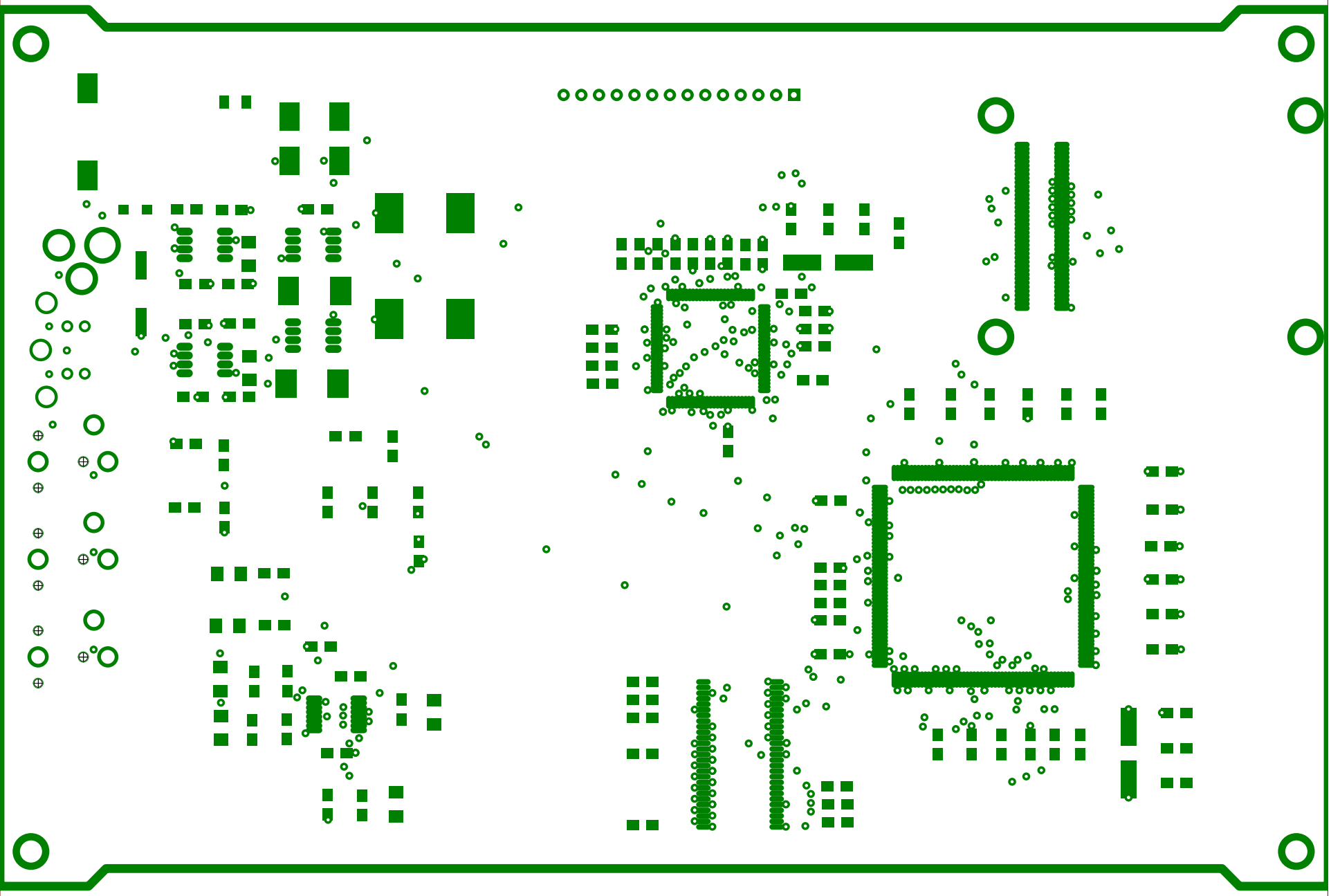
Last Modified: 12/8/2006

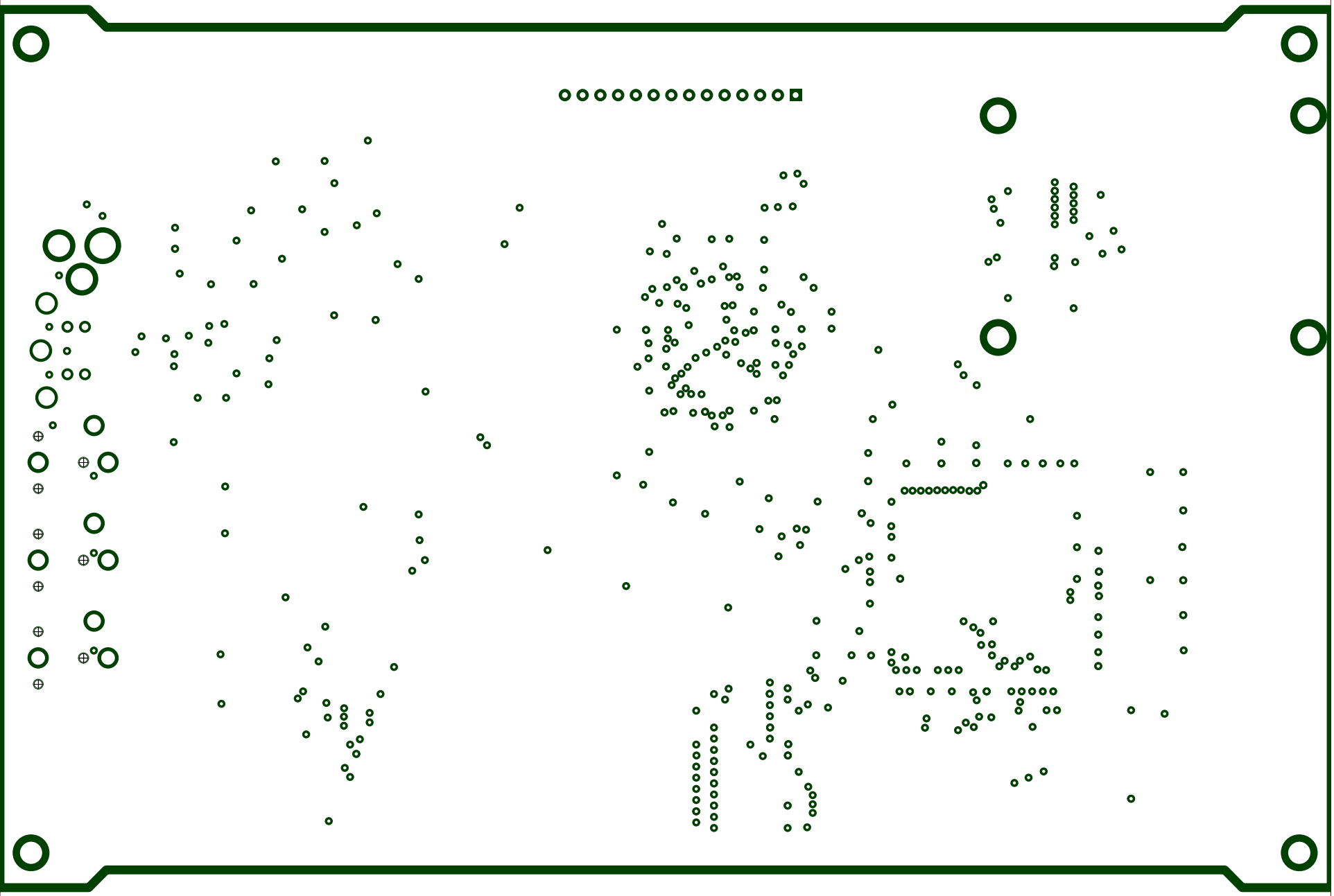
Created By: TEAM OMEGA CORE

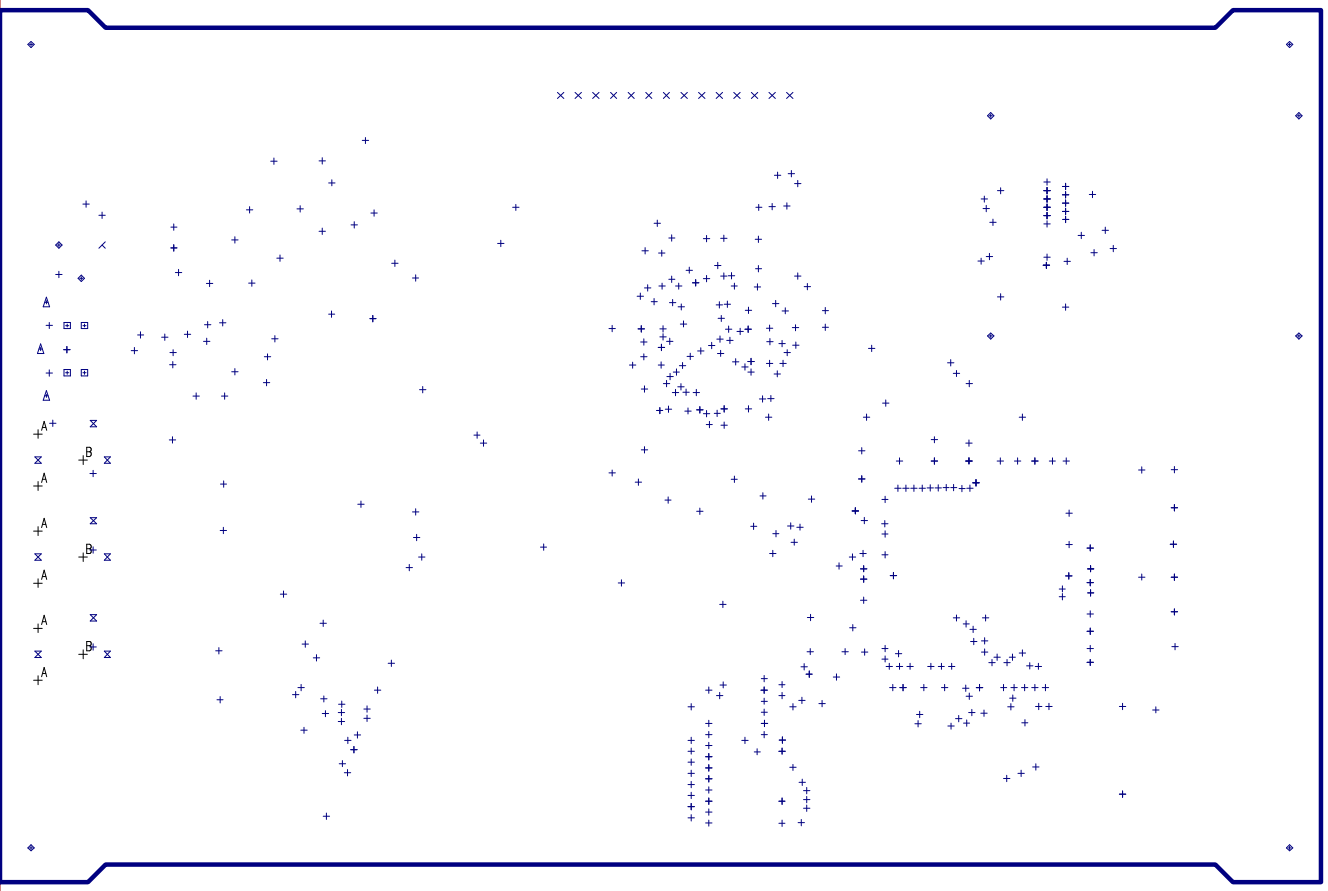
Matt Ricks, Irene Thompson, Eric Bowden











Drill Chart

Sym	Size	Qty	Tol	Type
+	0.02in	457	+/-0.004"	Plated
×	0.033in	14	+/-0.004"	Plated
⊞	0.04in	4	+/-0.004"	Plated
+A	0.0492in	6	+/-0.004"	Non-Plated
+B	0.052in	3	+/-0.004"	Non-Plated
⧖	0.079in	9	+/-0.004"	Plated
△	0.093in	3	+/-0.004"	Plated
◈	0.125in	10	+/-0.004"	Plated
⟋	0.15in	1	+/-0.004"	Plated

C64
C60
D4
C58
C62
DR125 Series
L3
DR127 Series
DR125 Series
L4
DR127 Series
D1
R33
R28
U5
U6
R29
D2
CON2
C63
C56
R32
R31
C57
CON1
U8
R30
D3
C61
C59
R3
R2
RCA1
R17
R16
R19
R18
RCA2
C17
R6
C16
R7
R8
RCA3
C18
R11
C20
R10
U2
C19
C21
R9
C22
C23
R13
R12
C24
C25
J1
C15
C7
C33
C3
C8
C31
C2
C6
C1
C13
L2
C14
C12
X1
C4
U9
R15
R20
R25
R21
R26
R22
R23
R24
U1
C5
C34
C35
C36
C37
C38
C39
C26
C47
C48
C49
C30
C27
U3
C50
R14
C28
C51
C29
C52
C40
C41
C42
U4
C43
C44
C45
C46
C71
C69
C68
C70
C67
C66
C65
R27
C54
C53
X2