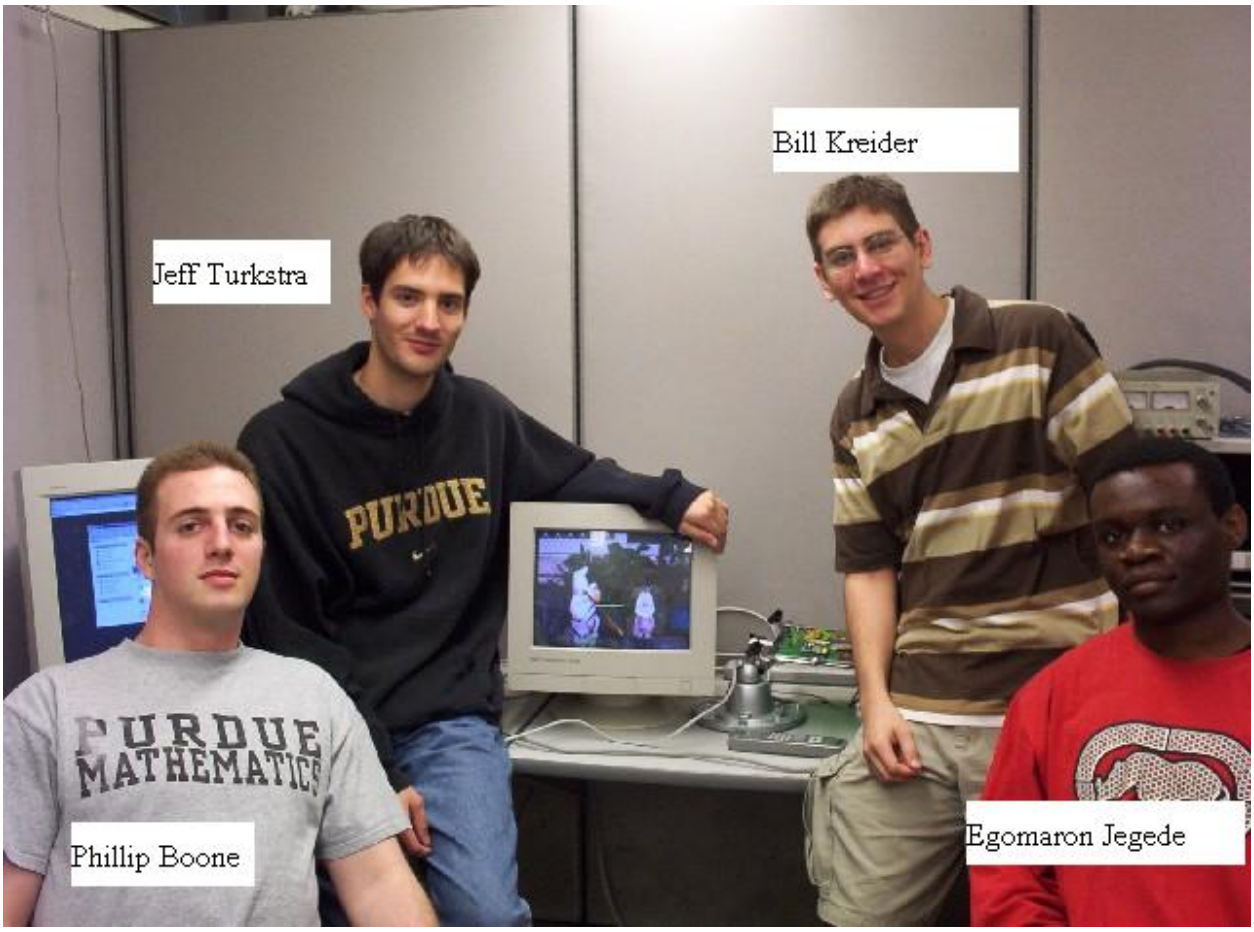
ECE 477 Final Report
Spring 2004



Team Code Name: ___Team _DiPFI_____ Team ID: __12__

Team Members (#1 is Team Leader):

#1: __Jeff Turkstra_____ Signature: _____ Date: _____

#2: __Bill Kreider_____ Signature: _____ Date: _____

#3: __Egomaron Jegede_____ Signature: _____ Date: _____

#4: __Phillip Boone_____ Signature: _____ Date: _____

REPORT EVALUATION

Component/Criterion	Score	Multiplier	Points
Abstract	0 1 2 3 4 5 6 7 8 9 10	X 1	
Project Overview and Block Diagram	0 1 2 3 4 5 6 7 8 9 10	X 2	
Team Success Criteria/Fulfillment	0 1 2 3 4 5 6 7 8 9 10	X 2	
Constraint Analysis/Component Selection	0 1 2 3 4 5 6 7 8 9 10	X 2	
Patent Liability Analysis	0 1 2 3 4 5 6 7 8 9 10	X 2	
Reliability and Safety Analysis	0 1 2 3 4 5 6 7 8 9 10	X 2	
Ethical/Environmental Impact Analysis	0 1 2 3 4 5 6 7 8 9 10	X 2	
Packaging Design Considerations	0 1 2 3 4 5 6 7 8 9 10	X 2	
Schematic Design Considerations	0 1 2 3 4 5 6 7 8 9 10	X 2	
PCB Layout Design Considerations	0 1 2 3 4 5 6 7 8 9 10	X 2	
Software Design Considerations	0 1 2 3 4 5 6 7 8 9 10	X 2	
Version 2 Changes	0 1 2 3 4 5 6 7 8 9 10	X 1	
Summary and Conclusions	0 1 2 3 4 5 6 7 8 9 10	X 1	
References	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix A: Individual Contributions	0 1 2 3 4 5 6 7 8 9 10	X 4	
Appendix B: Packaging	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix C: Schematic	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix D: Top & Bottom Copper	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix E: Parts List Spreadsheet	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix F: Software Listing	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix G: User Manual	0 1 2 3 4 5 6 7 8 9 10	X 2	
Appendix H: FMECA Worksheet	0 1 2 3 4 5 6 7 8 9 10	X 2	
Technical Writing Style	0 1 2 3 4 5 6 7 8 9 10	X 5	
CD-R of Website Image	0 1 2 3 4 5 6 7 8 9 10	X 2	
		TOTAL	

Comments:

TABLE OF CONTENTS

Abstract

The goal of this senior design project is the implementation of a digital picture frame interface (DiPFI). The central idea is the ability to interface between a PC acting as a network server and any VGA controlled display device effectively converting it into a digital picture frame. Challenging aspects of this project included implementing software for both the PC and the embedded micro-controller, addressing and sending data to the graphics controller and correctly interfacing between the various components.

On-device pushbuttons, status LEDs and a remote control were incorporated into the design, adding extra functionality and providing and intuitive means for the user to control the display. The DiPFI was successfully realized and pixel by pixel has written its way into digi-jock immortality.

1.0 Project Overview and Block Diagram

1.1 Project Overview

A Digital Picture Frame Interface, DiPFI, has been designed that will act as a buffer between a personal computer storing digital photographs and a VGA compatible display. JPEG images are decoded to raw .ppm files and then transferred to a Rabbit 3010 Core Module's[1] RJ-45 port. Once the data is received, the 21-bit memory address is sequentially latched into two Atmel 22V10 PLDs[2] operating at 3.3V. The PLDs' sole purpose is in Rabbit pin conservation, as only sixteen total pins are needed for both the 21-bit address bus and the 16-bit data bus. These PLDs latch the address so that the same Rabbit pins can be used to bus the data values to an Epson graphics controller[3]. The Epson has been equipped with a 4 MB EDO DRAM memory chip[4] that it uses to make the image data available for its onboard digital-to-analog converter. The Epson also controls the RGB outputs being sent to any VGA compatible display device. The system operates using a standard unregulated 9 VDC wall wart. The box also communicates with a standard Sony IR remote control, as well as onboard pushbuttons and status LEDs that enable the user to communicate with the device. The device is also driven using a 25.175 MHz external clock[5]. Power is generated by Low-Dropout Voltage Regulators[6]. The IR receiving is handled by a Sharp IR receiver[7] and data is decoded by a Reynolds Electronics Sony IR decoder[8].

1.2 Block Diagram

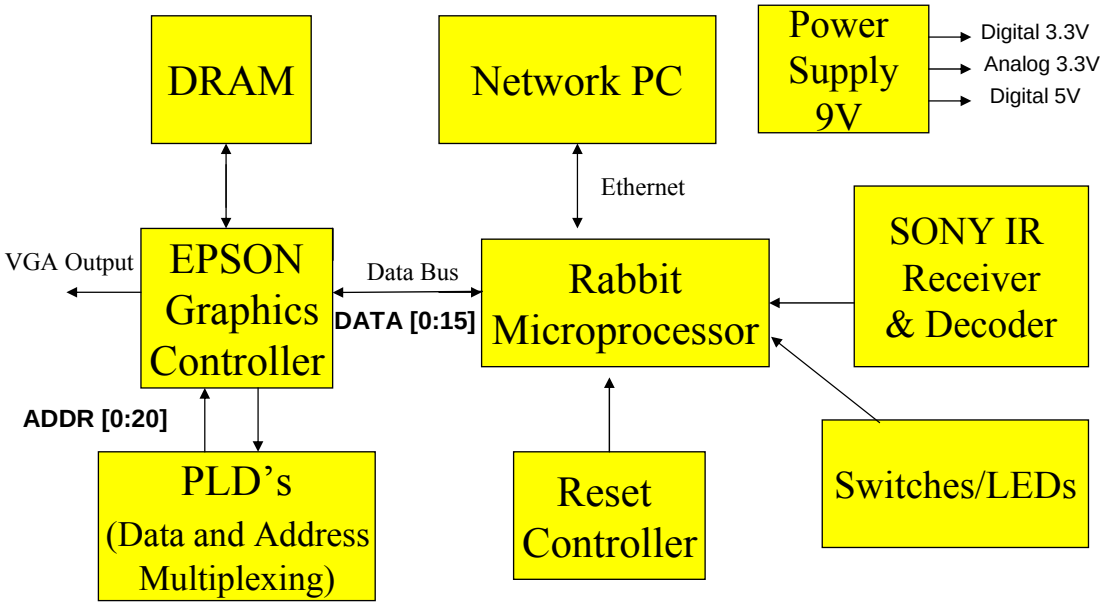


Figure 1-1. DiPFI Block Diagram.

1.3 Photograph of Completed Project

Figure 1-2. Completed Project.

2.0 Team Success Criteria and Fulfillment

2.1 Project-Specific Success Criteria

- Ability to interface to a VGA controller and display graphical data on a VGA device
- Ability to receive and interpret IR signals from a remote
- Ability to receive decoded image data via Ethernet
- Ability to interface with pushbuttons and status LEDs
- Ability to generate text overlay on VGA device

2.2 Evaluation of Fulfillment

- This criterion was the most challenging to fulfill and was working very well. There were, however, mapping issues because the Epson expected a 2 MB DRAM chip but saw a 4 MB chip. As such, the very top of the image was found at the bottom of the display screen.
- The criterion was working as expected. The IR signals were interpreted as expected and the picture displayed could be changed given the desired button press. There is little noticeable interference.
- JPEG images are decoded to .ppm files and then transferred via a TCP/IP client to the Rabbit. This decoded data is then forwarded to the graphics controller for processing, resulting in correct image display and complete fulfillment of the criterion.
- The onboard pushbuttons perform the same functions as the corresponding IR buttons and are interpreted as expected. The LEDs flash when busy and indicate when the device is receiving power, as expected.
- This criterion has yet to be fulfilled. Generating text-overlay on the Rabbit itself seems unnecessary and a software solution has yet to be created.

3.0 Constraint Analysis and Component Selection

3.1 Design Constraints

The two major constraints of the project are the micro-controller and the graphics processor.

The micro-controller must have a large amount of I/O pins if it is going to be interfaced with a graphics controller chip due to the size of the address bus and data bus of modern VGA graphics controllers. The micro-controller ideally should integrate as many features our design requires as possible. It should include a network adapter, extra I/O pins to interface to the graphics chip as well as the RF remote chip, have enough computational power to perform JPEG decoding, and finally an efficient development environment. The controller should have enough resources available to buffer the information coming from the Ethernet adapter and the data going to the graphics controller.

The graphics chip needs to have all of the necessary hardware to display an image on the VGA port. It will be responsible for creating the analog signals necessary to constantly refresh the image to the screen. It should have a frame buffer large enough to store a high-resolution image, so that once the image is sent to the controller it will be able to hold the image on the screen. If the frame buffer is external to the chip, it must have a dedicated interface to the graphics controller using readily available standard DRAM memory.

Finally, the picture box should also try and meet other design constraints. The Digital Picture Box is made to reside on a desk as a stand-alone device. Therefore it should be as aesthetically pleasing as possible. Portability is not a key issue, but the smaller and more lightweight the device is the better. The components should not cost more than an absolute maximum of a couple hundred dollars to be marketable.

- Large nuMBer of general I/O pins to interface with a graphics controller
- Integrated network adapter
- Efficient development environment
- Large amounts of processing power to decode JPEG and buffer video data

Table 3-1. Summary of Micro-controller Requirements.

- Ability to create all necessary analog RGB signals for a standard VGA adapter and refresh the image automatically
- Integrated frame buffer that is large enough to hold a SVGA image
- Standard EDO DRAM data bus for interfacing external memory chips

Table 3-2. Summary of Graphics Controller Requirements.

3.2 Rationale for Component Selection

The final decision for the graphics controller was narrowed down to the Epson S1D13505 "Embedded RAMDAC LCD/CRT Controller" and the Cirrus Logic CL-PS7500FE[9] "System-on-a-Chip with CRT/LCD Controller." Both chips have analog RGB outputs for a standard VGA CRT monitor, support for an external frame buffer using standard EDO DRAM chips, and a standard input bus for data transfer. The Cirrus Logic in addition supports a 32-bit ARM microprocessor, hardware floating-point unit, PS/2 serial ports, a 16-bit ISA bus, and serial CD quality digital sound. All of these features, which are not essential to the proposed criteria of the project, translate into the Cirrus Logic chip having almost twice as many pins as the Epson chip. Putting the overkill of features aside, the Cirrus Logic chip is not supported nearly as well as the Epson counterpart. The only document available for the CL-PS7500FE chip was a product bulletin that merely contains an overview of features, compared to Epson's 500+ page technical manual on their chip. Epson included timing diagrams for interfacing the chip with several different micro-controllers, pin descriptions, memory timing diagrams, and many other important details. There are also several documented student design projects on the Internet that successfully implemented the Epson controller.

It was considered very important that the project contain some type of Ethernet capability, so it seemed obvious to pick a variant of a Rabbit micro-controller with integrated 10Base-T Ethernet. Since the Epson controller requires 46 I/0 pins for its data and address busses alone, the RabbitCore 3010 is a perfect match. It contains 56 general-purpose I/O pins. This would leave 10 pins for the keypad interface and a few extra for debugging purposes. In order to maximize the speed that an image is loaded onto the display, all of the data should be sent to the graphics controller in a parallel fashion if possible. The Epson controller's RAMDAC operates at 40Mhz on a 16-bit bus. The Rabbit 3200 operating at 55Mhz should be able to keep pace with the Epson when transferring a high-resolution uncompressed image.

Epson S1D13505	Cirrus Logic CL-PS7500FE
● Analog VGA output ● External frame buffer using Standard DRAMs ● Digital input bus ● 128 pins	● Analog VGA output ● External frame buffer using Standard DRAMs ● ISA bus ● 240 pins ● 32-bit ARM microprocessor ● Digital serial sound output ● PS/2 Serial Interface

Table 3-3. Comparison of Epson and Cirrus Logic graphics controllers.

4.0 Patent Liability Analysis

Several patents were found that, at first glance, covered similar functions as those performed by the Digital Picture Frame Interface. However, upon further verification, it was found that only one patent poses any threat with respect to issues of patent liability. This device, marketed by Ceiva, Inc, consists of an LCD "frame" that dials in to a central repository and downloads images to be displayed on the device. This device performs a similar function to DiPFI; however, we feel that our device is an improvement because any used VGA device can become a digital picture frame.

4.1 Results of Patent Search

Upon researching patents at www.uspto.gov[10], the following patents described in Table 4-1 were deemed relevant to the product being designed.

Patent #	Description
6,442,573	*Method and apparatus for distributing picture mail to a frame device community* [15]
6,167,469	*Digital camera having display device for displaying graphical representation of user input and method for transporting the selected digital images thereof* [14]
6,111,586	*Electronic photo album editing apparatus* [13]
6,058,428	*Method and apparatus for transferring digital images on a network* [12]
6,037,989	*Still image transmitting device* [11]

Table 4-1. Patent #'s and Descriptions of Relevant Patents.

1. Patent 6,037,989 - This invention converts the image signal reproduced by a camcorder, TV, or VCR into serial data and transmits it to a personal computer as a still image signal, thereby making it possible to edit, store, or print the image signal, resulting in convenient use[11]. This device utilizes several transmit control signals to effectively transmit image data into a computer's serial port from an image sourcing device. Though this device transmits a still image to be displayed, which is similar in function to the DPB being designed, it involves transmitting data serially to a PC from another device. Therefore, the DPB does not infringe on this patent.

2. Patent 6,058,428 - Described here is a method and apparatus for transferring digital images on a network. A signature list is examined wherein the signature list includes a unique signature for each of the digital images requested for transfer[12]. The signature list is used to determine whether each of the digital images is present, and the images are checked for integrity. This provides an efficient means of tracking, recording, and processing image requests over a network. The DPB does not, however, infringe on this patent as the designers plan on implementing their own methods for sending images which do not involve the processes described within this patent.

3. Patent 6,111,586 - The apparatus described is an object to provide an electronic photo album editing apparatus which can easily edit an electronic photo album in compliance with a user's

various wishes[13]. This apparatus displays images stored on a personal computer to the computer's monitor in the form of a photo album giving the user a direct interface with which to edit said photographs and corresponding decorative elements. However, this is all generated in software and the DPB does not perform any digital signal processing on the given images as the box expects to receive raw 640 x 480 image data.

4. Patent 6,167,469 - In this patent, a method and apparatus for transporting digital images is described[14]. This apparatus pertains to the direct connection of a digital camera to a communications network for the transfer of said media. One embodiment of this system pertains to uploading the digital images to a website for direct display. The DPB connects to a network in much the same fashion; however, the patent pertains to the implementation within a digital camera, and does not relate to the image transfer being performed in the DPB.

5. Patent 6,442,573 - "A method and apparatus for distributing picture mail to a frame device community" is described. The present invention comprises one or more interconnected frame devices. Each frame device has a display region (e.g. an LCD) surrounded with a border region modeled to resemble a traditional picture frame. Each frame device is configured to connect to an interconnection fabric to periodically obtain image data from a centralized repository and then display that data according to criteria established by an authorized user. The data repository is populated with image data via the image collection process. Other information such as the behavior characteristics of each frame device are established and/or managed via a picture box. The picture box resides on a server computer and may be obtained by the user upon demand." [15] This patent very closely resembles the functions performed by the DPB, and thus it will be the main topic of discussion with respect to the patent liability analysis.

4.2 Analysis of Patent Liability

4.2.1 Literal Infringement

All aforementioned tasks involve the processing, transferring, and display of digital images, however none perform exactly the same function in exactly the same way as the DPB. Therefore, the DPB does not pose any literal infringement to any product currently on the market.

4.2.2 Doctrine of Equivalents

As was mentioned previously, Patent 6,442,573 was the only patent found that the DPB may infringe upon under the Doctrine of Equivalents. This product has been released by the company Ceiva located at www.ceiva.com[16]. The following characteristics of the patented device are similar to those devised for the DPB. The device described in this patent consists of a viewer or receiver that will be programmed to access a centralized repository via a telecommunications network using a modem or other network device (i.e. an Ethernet connection). The device is preprogrammed to access the repository to download and store any images uploaded to the server. The frame device comprises a CPU, memory, and telecommunications hardware. The inventors also describe what they call a "picture box," which differs completely from the DPB as defined here. The "picture box" they describe is a software utility that allows the user to configure specifications such as image update frequency for the frame device, and appears as a menu on the frame devices LCD screen. This is much like the text overlay that will be generated on the DPB. Data is also transmitted via picture mail and any image processing needed to fit the image to the display is done before transferring the data.

Unfortunately, the DPB performs, substantially, most of the same functions described previously in substantially the same manner. One difference is that the patented device contains its own frame device LCD display, whereas the DPB provides the internal circuitry and VGA output without the extraneous cost of the LCD display. Another difference is that the DPB does not store image data onboard other than the picture being displayed, while this device downloads and stores all pictures in internal memory. This allows the DPB to be as inexpensive as possible.

4.3 Action Recommended to Avoid Infringement

To avoid infringement, two courses of action can be taken. Unfortunately, it would be very difficult to adjust the DPB at this point in the design process so that it would not infringe on the item described in Patent 6,442,573. Therefore, the two options are paying royalties to obtain the rights for marketing the DPB or keep the DPB as a "personal project" that will only be used for recreational purposes. In keeping with the true spirit of the design project, the second option seems the most valid.

If any monetary payment was desired for this product, several steps would first need to be taken. First of all, a more exhaustive patent search performed by an independent firm would need to take place to determine if there are more potential infringements. If this search is negative, then a patent lawyer would need to be located to review the patents already located, especially Patent 6,442,573. If it were determined by a more credible source that the DPB indeed infringed on this patent, Ceiva would need to be contacted to find out what royalties need to be paid for the distribution of the DPB, and even this might not be sufficient to avoid legal action. This would be a very expensive process, and some indication that the DPB would be a commercial success would be desired before these actions would be pursued.

5.0 Reliability and Safety Analysis

Reliability issues pertinent to this project include component error, durability, emissions (heat and noise), speed and accuracy of image processing and image display. Safety issues were taken into account by securely casing the internal electronics and should not be a factor during normal operation. The aim of analyzing reliability and safety issues is to achieve the best performance with minimal cost and maximum customer satisfaction[17]. This is an important aspect of any products development process as it investigates any possible dangers or unwanted effects that may be experienced by the end user.

The reliability analysis is focused on 5 major design components that are most likely to fail due to frequent use, complexity or operating temperature and includes detailed calculations of failure rates for each component. In addition, a FMECA (failure, mode, effects, and criticality analysis) worksheet for the entire schematic grouped into functional blocks can be found in Appendix H.

The 5 design components with the highest probability of failure due to heavy use during operation are:

(i) Low Drop Out Voltage Regulators: 9–3.3V analog and digital and 9–5V digital
(ii) Rabbit 3000 microprocessor
(iii) IR receiver/decoder
(iv) EPSON graphics controller
(v) DRAM memory chip.

Analysis and calculations are done with reference to the formulas and variables in the Military Handbook Reliability Prediction of Electronic Equipment[18].

Failure rate calculation variables :

MTTF: Mean time to failure $(\lambda_p)^{-1}$

λ_p : represents the predicted nuMBer of failures per 10^6 hours of operation.

λ_{BD} : die base failure rate

λ_{BP} : package base failure rate

λ_{EOS} : electrical overstress failure rate

λ_{cyc} : cycling factor

C_1 : die complexity constant

C_2: pin nuMBer constant

π_T : temperature coefficient, based on junction temperature

π_E : environmental constant, based on equipment use environment

π_Q : quality factor, military (1-2) or commercial 3 -10

π_L : learning factor, based on years device type has been in production

π_{MFG} : manufacturing process correction factor

π_{CD} : die complexity correction factor

π_{PT} : package type correction factor

(i) LDO Voltage Regulator : 9V – 3.3V analog (Texas Instruments REG103-33)

Linear MOS Device $\lambda_p = (C_1\pi_T + C_2\pi_E)\pi_Q \pi_L$

Parameter	Value	Justification
C_1	0.02	Linear, 101–300 transistors
π_T	7.0	Linear, MOS, $T_J < 85^0$ C (assuming max temp.)
C_2	0.0025	SMT, 6 functional pins, non-hermetic
π_E	2.0	Ground fixed environment
π_Q	10	Commercial
π_L	1.0	Years in production > 2.0

Table 5-1. LDO Voltage Regulator Calculations.

$\lambda_p = (0.02*7.0 + 0.0025*2.0)10*1.0 = 1.45/10^6$ hours

MTTF = $1/\lambda_p$ = 689 655.17 hours = 78.73 years

The 9V-3.3V digital and 9V-5V digital belong to the same LDO regulator family of components from the same manufacturer and have similar operating temperatures hence the above calculations are applicable in determining failure rates for those components.

(ii) Rabbit 3000 microprocessor

Digital MOS Microprocessor with > 60000gates $\lambda_p = \lambda_{BD}\pi_{MFG}\pi_T\pi_{CD} + \lambda_{BP}\pi_E\pi_Q\pi_{PT} + \lambda_{EOS}$

Parameter	Value	Justification
λ_{BD}	0.16	Logic device
π_{MFG}	2.0	Non QML or Non QPL
π_T	0.88	Digital MOS, $T_J < 85^0$ C (assuming max temp.)
π_{CD}	25	1.6 cm^2, assume 1 micron size
λ_{BP}	0.0044	128 pins
π_E	2.0	Ground fixed environment
π_Q	10	Commercial
π_{PT}	6.1	Non-hermetic, surface mount
λ_{EOS}	.065	Max voltage is 5.5v , ESD Susceptibility

Table 5-2. Rabbit 3000 Microprocessor Calculations.

$\lambda_p = 0.16*2.0*0.88*25 + 0.0044*2.0*10*6.1 + 0.065 = 7.6418/10^6$ hours

MTTF = $1/\lambda_p$ = 130 859.22 hours = 14.938 years

The rabbit microprocessor is being used as part of the core module which includes additional components (SRAM, Flash and Ethernet) which will negatively affect its overall reliability and performance.

(iii) 14-bit IR Decoder (IR-D14 IC)

Digital Microcontroller $\lambda_p = (C_1\pi_T + C_2\pi_E)\pi_Q \pi_L$

Parameter	Value	Justification
C_1	0.02	MOS Digital, assume < 1000 gates
π_T	0.88	MOS Digital , $T_J < 85^0$ C (assume same max temp)
C_2	0.0034	SMT, 8 functional pins, non hermetic
π_E	2.0	Ground fixed environment
π_Q	10	Commercial
π_L	1.0	Years in production > 2.0

Table 5-3. 14-bit IR Decoder Calculations.

$\lambda_p = (0.02*0.88 + 0.0034*2.0)10*1.0 = 0.244/10^6$ hours

MTTF = $1/\lambda_p$ = 4 098 360.65 hours = 467.84 years

(iv) EPSON Graphics Controller (S1D13505)

LCD/CRT Controller $\lambda_p = (C_1\pi_T + C_2\pi_E)\pi_Q \pi_L$

Parameter	Value	Justification
C_1	0.08	MOS Digital, assume 30,001 to 60000 gates
π_T	0.88	MOS Digital , $T_J = 85^0$ C (max operating temperature)
C_2	0.068	SMT, 128 functional pins, Non-hermetic
π_E	2.0	Ground fixed environment
π_Q	10	Commercial
π_L	1.0	Years in production > 2.0

Table 5-4. EPSON Graphics Controller Calculations.

$\lambda_p = (0.08*0.88 + 0.068*2.0)10*1.0 = 2.064/10^6$ hours

MTTF = $1/\lambda_p$ = 484 496.12 hours = 55.31 years

(v) 16-Bit EDO DRAM chip.

MOS Memory Device $\lambda_p = (C_1\pi_T + C_2\pi_E + \lambda_{cyc})\pi_Q \pi_L$

Parameter	Value	Justification
C_1	0.01	DRAM, Memory size > 256K
π_T	5.0	Memories , $T_J < 85^0$ C (max operating temperature)
C_2	0.019	SMT, 40 functional pins, Non-hermetic
π_E	2.0	Ground fixed environment
λ_{cyc}	0	Non EEPROM device
π_Q	10	Commercial
π_L	1.0	Years in production > 2.0

Table 5-5. 16-Bit EDO DRAM Chip Calculations.

$\lambda_p = (0.01*5.0 + 0.019*2.0 + 0)10*1.0 = 0.88/10^6$ hours

MTTF = $1/\lambda_p$ = 1 136 363.63 hours = 129.72 years

Summary and Conclusions

Component	Description	$\lambda_p / 10^6$ hours	MTTF years
U14	LDO voltage regulator (REG103-33)	1.45	78.73
R1*	Rabbit 3000 Microprocessor	7.64	14.94
U16	IR Decoder	0.24	467.84
U10	EPSON Graphics Controller	2.06	55.31
U9	EDO-DRAM	0.88	129.72

Table 5-6. Component Failure Rate Summary (R1* Only rabbit headers are on schematic).

The average overall failure rate is λ_p : 2.454 / 10^6 hours or a MTTF of 149.308 years.

As shown above in Table 5-6 among the 5 components analyzed the Rabbit Microprocessor is the most likely to fail by a large margin. Given that all calculations were made using a maximum operating temperature of 85^0 C as compared to the standard 25^0 C these error rates are higher than what would occur during normal operation of the circuit. It is still valuable however to reduce error rates as much as possible and this could be achieved by adding a heat sink to the Rabbit, EPSON and the LDOs which have the highest operating temperatures thus extending the theoretical lifetime of the digital picture box.

FMECA (Failure, Mode, Effects and Criticality Analysis)

In conducting the FMECA Analysis, the DiPFI schematic is divided into its major functional blocks:

Blocks	Category	Components
A	Power Supply	9V Wall wart, Voltage regulators
B	User Interface	Push buttons, IR Decoder and Receiver
C	Physical connectors	VGA Connector, RJ-45 (on core module)
D	Graphics	EPSON, DRAM, PLDs , Crystal Oscillator
E	Controllers	Rabbit 3000 (Headers), Reset controller

Table 5-7. Major Schematic Categories and Components.

See attached worksheet in Appendix H for detailed analysis of all possible failure conditions for each block, the resulting effects on other parts of the design and the level of criticality for each type of failure.

There are two basic levels of criticality as pertains to this project:

LOW - meaning the overall output of the design will not be impacted and a display is still visible on the display device.

HIGH - meaning the output will be affected and the design not function as expected. Specifically a distorted image or no image at all present on the display device.

For LOW criticality failures a rate of $\lambda < 10^{-4}$ will be accepted and for HIGH criticality failures a rate of $\lambda < 10^{-9}$ errors per hour of operation.

6.0 Ethical and Environmental Impact Analysis

As with any engineering project, there are numerous ethical and environmental concerns that should be addressed. In particular, since this project utilizes solder and a printed circuit board, there are almost certainly some environmental concerns. Those aside, it's imperative to provide the end-user with a product that is safe, usable, and reliable. All of these have ethical implications that lie beneath them.

6.1 Ethical Analysis

There are numerous ethical issues related to this particular project if it were to actually go into a manufacturing stage. The first, and most important, is user safety. Since the device would ultimately be enclosed in a "box" of some sort, permitting the user only to press buttons and utilize the IR remote, many of these issues resolve themselves. However, there are still a few basic issues that should be addressed. Care should be taken to organize the components in such a manner as to minimize the risk of a short or some other failure related to the box being moved or shaken (during an accidental fall, for instance). That aside, care should also be taken to ensure that the box is grounded (if it or portions of it are made of metal) and that it isn't possible for the user to come into contact with any circuitry while pushing the buttons on the device. The device is not designed to have a normal user service its internal parts. As such a warning label should be clearly placed somewhere indicating that there are no user-serviceable parts inside and that the user risks electrocution by opening the box. Once that is accomplished, there should be no ethical liability (or legal liability, for that matter) with regards to a user attempting to service components internal to the device. Finally, as a form of extra precaution, it would probably be desirable to build some type of current monitoring device into the system to detect any shorts (if, for instance, the device has some type of liquid spilled on it) and immediately terminate power.

Next up is reliability, something that can easily destroy a product's value if it doesn't exist. The device should be thoroughly tested in various operating conditions before being placed into production. These conditions, of course, should reflect the intended environment…namely indoors. Tests should include, but are not limited to, reliability (how long individual components last), durability (how does the product deal with every-day "abuse"), and finally safety (what happens if the device is repeatedly dropped, submerged in water, etc). Consideration must also be made for the remote control and 9V wall wart. If appropriate target results aren't met, it would be prudent to select more reliable parts at that point or perhaps establish and notify the

user of the device's life expectancy. As briefly touched upon at the beginning of this paragraph, potential fallout from having an unreliable product includes numerous things. The device would quickly gain a poor reputation; unforeseen problems could hypothetically result in some sort of harm to the user (a seizure from a certain screen refresh frequency, for instance); and of course product sales would decrease significantly.

Finally in terms of ethics (and a quality product, for that matter) comes usability. This particular device is fairly intuitive, although consideration should be given to the onscreen menu system to ensure that it is easy to understand and efficient. Another issue which may not be readily apparent is the fact that this is a networked device. In other words, network security should also be ensured. Some type of authentication clearly needs to be added to ensure that only proper users can obtain and manipulate control of the device. In terms of actual data security, since it is assumed that the images will be displayed in a somewhat public setting anyway, data security (i.e., encryption) is most likely unnecessary. Without these additional insurances, the same problems mentioned above could again be encountered; namely, the device gaining a poor reputation and the ultimate drop in sales revenue.

Ultimately ethics play a large roll in the revision process before this product goes into actual production. It is clear that the existing prototype is unfit for public use, and to ensure a quality product, capable of turning a profit, numerous additional features should be built-in.

6.2 Environmental Analysis

Ethical considerations aside, there are also environmental implications related to this device. At first thought this may not be obvious, since many people consider digital systems to be incredibly clean and efficient devices, at least with respect to "traditional" pollution. However, this particular device has an environmental impact through all stages of its life; from manufacture, to normal usage, and ultimately disposal/recycling.

The manufacturing process realistically has the most potential to be environmentally damaging. Traditional PCB fabrication processes tend (at least until recently) to use lead-based solder which besides the obvious fact that it contains lead, also tends to release toxic fumes when heated. With respect to this precautions need to be taken to properly contain and vent the fumes as well as control the amount of lead released into the environment. This can be easily resolved by using solder that isn't lead based, but in some cases this can be more expensive. Printed circuit boards contain amounts of lead as well and at this point there a real cost-effective method

around this fact is lacking. However, the manufacturing processes are mature enough that the environmental impact is minimal. However, as we will see shortly, it does present some concerns during device disposal. In addition to these potential environmental hazards, one also has to consider the manufacturing process behind each IC used as well as other components. Each has an associated environmental "cost."

In terms of normal usage, the only notable environmental impact would be with respect to the energy band (i.e., radio waves, IR waves, etc) and energy usage. The device does utilize IR signals and has potential to interfere with similar devices. Moreover, since this is a digital device with switching logic it has the ability to generate its own electromagnetic field which could easily interfere with other nearby devices. Overcoming this can be done through sane design layouts as well as constructing the external casing of a material that would assist in reducing EMF output. It is important to note that realistically only the former is a viable solution, as the FCC generally frowns on reducing EMF just by throwing the device into a Faraday's Cage. Finally, something that is often overlooked is the device's power consumption. Despite what many believe, electricity is not in infinite supply. Moreover, many of the techniques used in electricity generation are still taxing on the environment. As such, careful consideration should be given to selecting low power devices and minimizing any potential extraneous energy usage. These aside, the device is fairly benevolent during its operational life, at least with respect to the environment.

Of course, the end of a products life cycle can have almost as much of an impact on the environment as its start. As mentioned earlier, PCB's contain lead, which by itself essentially mandates that instructions be included on proper disposal of the device. It should not simply be thrown into the garbage. The best course of action would be to take the device to some type of recycling plant equipped to handle PCB's and related circuitry. Here parts may potentially be recycled and it can be ensured that the lead containing materials are handled properly.

As one can see, device design and manufacture is composed of more than simple "engineering" concerns. There are numerous ethical and environmental issues that are at the heart of a product's design as well.

7.0 Packaging Design Considerations

Packaging Design Considerations

Physical features of the DiPFI include power and status LED's to indicate the device is on and that a picture is being transferred respectively. Push buttons are used to cycle through the pictures stored on a remote PC. The device will be capable of being controlled remotely with the same functionality as the push buttons. The DiPFI design layout consists of a light weight plexi-glass casing with push buttons and LED's on the front and the following connectors on the rear: VGA connector, RJ-45 network connector and 9V power connector.

In the following sections all of the packaging requirements pertaining to the DiPFI are discussed. A detailed analysis of similar commercial products is included. Specifications for the packaging of the DiPFI along with a detailed rendering illustrating the shape and size are in Appendix B. A materials list of components needed for the packaging and an approximation of the DiPFI weight and unit cost is included in Appendix E. A list of references to the commercial products considered can be found in Section 13.0.

Commercial Product Analysis

The digital picture box can be thought of as a modular separation, in terms of function (information processing/decoding/transmission), from the display component of the many CPU enabled or stand alone digital picture frame devices on the market. The DiPFI evolved from that idea hence a product analysis of a digital picture frame is included. Our design aims to be more flexible, the separation enabling the DiPFI to operate with any VGA display screen or monitor.

The first commercial product analyzed is the Digi-Frame DF-1710[19] shown below in Figures 1 & 2. This high definition display in a natural wood frame is wall mountable and loads pictures via CD-ROM, another PC or the internet via Ethernet. It can show JPEG/MPEG-1 content, play MP3's and has many additional features for storage, security, and remote control. The dimensions of the frame are (WxHxD): 17.83" x 14.5" x 2.9" (23.5" x 19.5" x 3.25" with frame) with a screen size of 13.38" x 10.58" and weight of 19 pounds.

Figure 7-1. Digi-Frame Front. **Figure 7-2. Digi-Frame Back.**

The most impressive aspect of this product is that so much functionality is hidden behind the XGA-resolution display adding a depth of only 2.9" to the frame. The size of the display is striking and with the natural wood frame is an appealing package. In addition the packaging allows the frame to be rotated vertically allowing the customer to choose their preferred placement.

The main disadvantage of such a large display and frame, (24" x 20") is the lack of portability. Once placed or mounted this product can't easily be moved due to its weight and size and the sensitivity of the CD-ROM drive is an additional concern.

Our design packaging implements the same RJ-45 connector allowing picture transmission from the internet but adapts this interfacing to enable output to any VGA monitor or LCD display. The fundamental difference in packaging that makes our digital picture box unique is that the image processing is separated from the display allowing the consumer to switch the display with ease. The small size of our product means it is portable and allows it to be placed on a desk, beside or even out of sight behind the desired display.

The second commercial product being analyzed, the Vosonic Multi Media Viewer[20] has slightly differing functionality but shares many of the same packaging considerations and goals as our device.

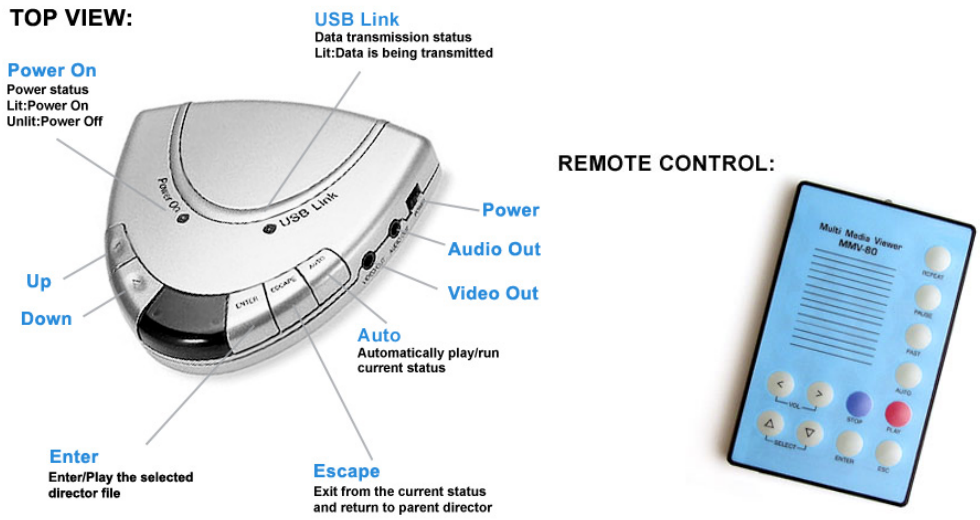


Figure 7-3. Vosonic MMV-80 (Dimensions: 95 x 89 x 15 mm - Weight: 81g.).

The MMV plays JPEG/MPEG/MP3 file formats, supports various memory cards is compatible with Windows 98/SE/2000 and Mac OS v8.6+ with USB driver and can play these files through NTSC/ Pal TV's or TFT monitors. As shown above, it can be controlled remotely with an IrDA remote.

The MMV has a compact and sleek design with an easy to understand operation keypad making it appealing to consumers as it is portable and can sit atop or beside whatever screen is being used for the display. The outputs and inputs are well arranged on the sides of the casing and the LED's positioned on top where they can clearly be viewed. In addition the panel for IR signal reception is prominently placed ensuring a wide field of sight for the remote control operation in a room.

The most desirable aspect of this packaging is the user friendly and compact design which makes good use of space without cluttering or being too small for easy use. The buttons are evenly spaced and labeled as to their function making the product easy to understand and operate. These are features we incorporate into the design of our DiPFI packaging. With careful and creative planning we may be able to achieve a more compact design due to the fact that the DiPFI will not have a memory card slot and will possess only one 15 pin female HD connector as a video out. In addition the four pushbuttons (left, right, function and power) are the only functions needed on the remote control hence a smaller sleeker version will be used in our

product. We use similar plastic casing that is lightweight but tough to make the picture box sturdy and easily portable.

8.0 Schematic Design Considerations

8.1 Theory of Operation

8.1.1 Rabbit 3010 Core Module

The Rabbit 3010 best suites the need of this project because of its Ethernet capabilities as well as its large nuMBer of I/O pins. The Rabbit has a maximum internal clock frequency of 29.4 MHz generated by a 14.7456 MHz crystal. The Rabbit boasts an internal clock doubler that allows it to achieve the maximum clock frequency of 29.4 MHz. Therefore, the graphics controller is clocked externally using a 25.175 MHz clock, while running the Rabbit at its maximum clock rate of 29.4 MHz. This clock frequency gives a ~183 kbps baud rate, which is sufficient for the needs of the project. The Rabbit takes a regulated 3.3 VDC power supply. To achieve this from a 9 V unregulated power source, a 500 mA Low-Dropout Voltage Regulator[6] is used to drop the 9 V to a regulated 3.3 V. The RCM 3010 also contains 128 kb of SRAM which is adequate for software storage and any picture buffering as needed.

Another consideration on the Rabbit is how to interpret the image data packets arriving on the Ethernet port. The PC connects to the Rabbit controller using TCP/IP and begins sending data pixel by pixel. Each packet contains address and data information. When the packet arrives at the Rabbit port, the packet is read and the data dropped into a buffer. Another routine works on forwarding any information contained to the graphics controller. Because of a lack of input pins, the address is loaded into two Atmel 22V10 PLDs. These were chosen because of their operation at 3.3V, as well as their DIP package. The PLDs take two mode pins for a total of four states. One mode latches the lower 11 of 21 address bits into one PLD. The second mode latches the rest of the address bits into PLD number two. The last mode enables the data bus and begins communications with the Epson Controller. Since there are eight control signals on the Epson, a third PLD is used to multiplex the signals as necessary.

8.1.2 Epson Graphics Controller

With 130+ pins, the Epson graphics controller poses a major interfacing concern. It has 21 address pins, 16 data pins, 9 memory address pins, and 16 memory data pins. It also contains 5

pins that interface to a standard VGA connector for picture display. As aforementioned, this interfacing is handled with a minimum number of PLDs and control signals. The Epson data sheets do not stress any timing constraints, so this will be a sufficient method for transmitting the image data.

Another concern on the Epson is its onboard Digital to Analog converter, which requires a separate 3.3VDC power and ground from that of the digital circuit. We accommodated this using a separate low-dropout regulator and ground. The DAC also requires a 4.6 mA current reference, IREF, which is provided by a 2n2222 NPN transistor.

8.1.3 Sharp IR Receiver[7] and Reynolds Electronics Decoder[8]

The Sharp IR receiver operates at 40 kHz, which is the modulation frequency of the standard Sony remote control protocol. This way, any Universal Remote can be used to control the device. An issue arrives with unwanted interference from other Sony-type remotes, in that the device is not intelligent enough to differentiate between two remotes transmitting with the same protocol. A solution has not been conceived, and it is not a priority at this time.

A Reynolds Electronics IR decoder chip was found that inputs the received IR signal and toggles a corresponding output pin. This works for buttons 0-9 and the channel and volume buttons. The channel and volume buttons are utilized in this implementation. The Rabbit monitors its input pins and watches for a toggled bit at the decoder output. It then executes the appropriate command. The Rabbit can sink up to 6 mA of current, so the output current of the decoder chip was monitored early on to ensure compatibility with the Rabbit.

An IR decoder that operates at 3.3 V could not be found. As such, a final low-dropout regulator is used to achieve a digital 5 VDC power supply. Also, the Rabbit can handle a maximum 5.5 VDC at its input pins, thus enabling the direct connection of the decoder to the input pins of the Rabbit without using level translation.

8.1.4 LEDs and Pushbuttons

There is little concern as far as power/current limits here. Two different colored LEDs, green and yellow, are used to indicate status signals, blinking when an image is being transferred. The source/sink limit for a Rabbit at 29.4 MHz is 6 mA, so a current limiting resistor has been used.

Two Rabbit input pins are used to toggle the LEDs state. The LEDs can be shut off using the Hi-Z state of both input pins, but there was no need for this in the current implementation.

8.1.5 Clock

A 25.175 MHz oscillator is used as a separate means to externally clock the Epson Graphics Controller.

9.0 PCB Layout Design Considerations

In order for the aforementioned components to work properly together, careful attention must be given to the printed circuit board layout. The fact that the Epson contains a DAC, for instance, affects layout strategies considerably. The remainder of this section focuses on what exactly these concerns are as well as the approaches used to overcome them.

As mentioned in the Motorola Semiconductor Application Note, one of the most important concerns for layout is the power system. Specifically, keeping components that are sensitive to noise (analog devices, for instance) separate from those that create noise (i.e., high power devices, fast switching devices, etc). In the digital picture box's case there are two primary components that need to be kept separate from each other: the digital part of the board and the analog DAC present on the Epson graphics controller. This separation is achieved through a couple of ways. First, the DAC has a separate LDO for its power source. This, coupled with the separation of DAC and digital ground via a ferrite bead, eliminates the ability of digital noise to be directly introduced into the DAC.

At this point there are still indirect concerns; namely, coupling through magnetic radiation (i.e., nearby digital components and their traces affecting the DAC through EMF coupling). A few steps were taken to prevent this. First, all analog components were kept a notable distance from other digital components. Second, in most cases traces that had to run over or under a trace relating to the analog part of the system were run in a manner that made them perpendicular to the analog traces, thus reducing the EMF coupling. Finally, the +5V LDO and related components were placed on the opposite side of the board, in an attempt to keep the relatively higher power aspects of the system as far away from the DAC and analog components as possible.

In terms of the general board layout, numerous methods were employed to reduce EMI and related noise. For starters, traces run on the bottom layer of the board were done from top-to-bottom in most cases whereas traces run on the top layer were done from left-to-right. Again, as mentioned above this was done to reduce the EMF coupling and induced noise between traces. The notable exception to this is the region surrounding the Epson controller. Given the roughly 6.5mil sized pads and relatively small spacing constraints, many of the traces on the top layer ended up running from top-to-bottom. Next, decoupling capacitors were placed as close to the IC's as possible to reduce noise and compensate for current spikes related to unexpectedly high transistor switching. Decoupling capacitors were generally placed directly beneath the main IC's (which consist of the Epson controller, the EDO DRAM chip, the IR controller, and LDO regulators). Finally, per the Motorola document's recommendations, the 25.125MHz crystal was also placed as close to the Epson (the only externally clocked device onboard) as possible. Clock traces were kept as short as possible and any inductive loops were avoided at all costs.

EMI aside, component placing was also influenced by actual components' purposes. The VGA connector was placed on a board edge, as would be expected. The IR receiver was also placed on an edge in addition to the push buttons and LED's. The rabbit's built-in Ethernet jack is not on an edge for routing simplicity. Instead, we intend to connect an "extender dangle" to the rabbit and run it outside of the actual box.

Priority was also taken into consideration during the routing process. The Epson, being the smallest and most complicated chip, was routed first; followed by the DRAM, the Address and Control PLD's, the Rabbit microcontroller, LED's and Pushbuttons, "Analog" components, "IR" components, and finally the debug headers.

Most trace sizes were 12 mils with the exception of traces related to the Epson. Larger trace sizes were attempted for power, but generally caused problems in most areas. As such, 12 mils is used consistently throughout the board in most cases. 90 degree angles were also avoided whenever possible.

All of these considerations and other minor ones were combined and utilized in what hopefully turns out to be a successful attempt to create a usable and environmentally-friendly PCB layout.

10.0 Software Design Considerations

The Rabbit 3000 micro controller has several general design considerations. Since the controller comes with a Dynamic C compiler, it automatically handles the memory mapping and management at compile time. The compiler also sets up the necessary startup code so the downloaded program will automatically execute on power up. There is also support for a simple multitasking environment included with the compiler. The current proposed implementation for sending address information to the Epson controller is somewhat inefficient. Transferring a single picture could take a fair amount of time. It was decided that in order to create a responsive software interface, we should take advantage of the built in multitasking functions to easily interrupt a picture transfer in progress. There are three basic C functions that implement a cooperative multitasking environment. First, a "costatement" designates a piece of code to a single running process. Within that process, it has the option to call the "yield" function. This gives up control to the next process. When the process that called yield obtains control again, it returns to the line of code after the yield function. The "abort" function is very similar to yield; however, when the process regains control it starts execution at the beginning of the costatement block.

10.1 Software Design Narrative

In order for the PC program module and Rabbit module to successfully communicate together, they must obey some type of protocol. It was decided to implement a custom TCP/IP Picture Frame Protocol. The packet data that is sent from the Rabbit to the PC file server is just plain text ending in a null terminating character. The Rabbit only needs to send four commands to the PC: "nextpic," "prevpic," and "recv." "Nextpic" tells the PC to change its current file pointer to the next available picture file. "Prevpic" tells the PC to change its current file pointer to the previous picture file. "Recv" tells the file server to begin sending the current picture file. The packet data that is sent from the PC to the Rabbit requires the use of two length fields. The Rabbit dumps all of the incoming data into a temporary buffer. A total length field that is placed at the beginning of each packet is necessary to distinguish between consecutive packets dumped into the buffer. Once a single packet is extracted from the buffer, it is necessary to separate the included command from binary data. The second part of each packet is a command length integer, which is the

nuMBer of bytes the included command occupies. There are also four commands sent to the Rabbit: “sendpicname,” “sendnewpic,” “sendnextpixel,” and “sendpixeladdress.” “Sendpicname” includes a text string that is the filename, as it is stored on the PC. “Sendnewpic” resets the current address pointer to the frame buffer on the Epson graphics controller to the first address, and sends the first pixel’s color data. “Sendnextpixel” increments the frame buffer address pointer, and sends the corresponding color data for that pixel. “Sendpixeladdress” sets the frame buffer address pointer to an absolute address, and then sends the color data for that pixel.

Incoming Rabbit packets	Incoming PC packets
* Packets can include Commands and Data	* Null terminated commands
* Total length used to distinguish between consecutive packets stored in the Rabbit's temporary buffer	* Commands Include: getpicname, nextpic, prevpic, and recv
* Command length used to separate the Command from Data	
* Commands include: sendpicname, sendnewpic, sendnextpixel, and sendpixeladdress	

Table 10-1. Summary of TCP/IP Communication Protocol.

The Rabbit program consists of several programming modules, which are actually costatements that implement the cooperative multitasking environment described earlier.

The first costatement constantly runs the tcp_tick command. This is done to perform all the necessary low-level TCP/IP bookkeeping.

The second costatement “incoming_tcp” starts listening on port 39 for any TCP/IP connections. Once it detects a connection, it takes all incoming data, and drops it into a temporary circular buffer called “indata.” If “indata” is full, it yields control until there is enough space to put the data into the circular buffer. This is repeated while the PC is connected. Once the PC disconnects, the Rabbit goes back to listening on the network port.

The third costatement “monitorIR” polls the IR input pins two samples at a time. When an IR remote button is pressed, the logic value on the output of the IR decoder pin changes. This can be detected when two consecutive samples are opposite logic values, indicating a recent logic change. Whenever a button press is detected, the Rabbit sends the corresponding command to the PC if a network connection is present.

The fourth costatement “monitor_buttons” is the same as “monitorIR,” except it debounces the switch inputs and sends the corresponding command to the PC if there is a network connection. The switches are debounced by taking three samples of each input pin. There is a delay of 50 ms inserted in between each sample via the “DelayMs” function that automatically configures a timer interrupt. Once three samples agree, the logic value of the samples is compared with the previously obtained agreeing samples. If a logic value zero was followed by a logic value one, indicating a rising edge, then a button press was successfully detected.

The fifth costatement “decodePCdata” consumes data contained in the circular buffer one command at a time. Each decoded command is stored in a temporary location. The command length is read in, and then the entire command is read in if possible. If it is not possible to decode an entire command, control is yielded to the next process. Once the command is decoded, the proper address is sent to the address bus of the PLDs that interface with the Epson graphics controller. The address is clocked in 8 bits at a time for three clock cycles until the entire address is present. The 15 bit color data is then sent out on the data bus and the Rabbit generates the necessary signals to load the current pixel into the frame buffer of the Epson graphics controller.

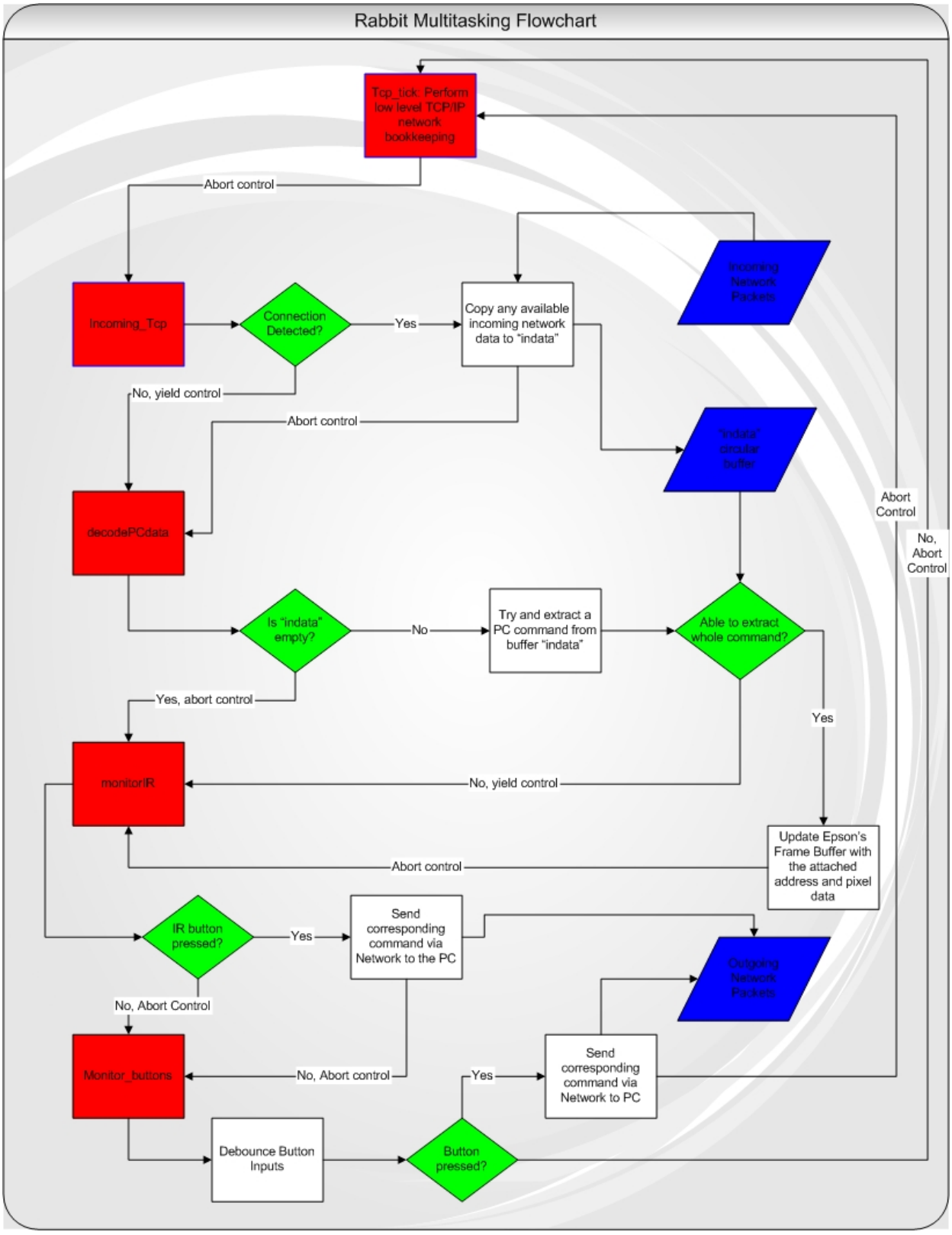


Figure 10-1. Flow Chart of Rabbit Programming Module.

Programming Module	Description
tcp_tick	Constantly called to perform all necessary low-level network bookkeeping.
incoming_tcp	Listens to the network port. When connected, it drops all incoming data to a circular buffer if possible.
monitorIR	Monitors the infrared port. If a command is detected, the corresponding command is sent via TCP/IP to the PC.
monitor_buttons	Monitors the button input. If a command is detected, the corresponding command is sent via TCP/IP to the PC.
decodePCdata	Decodes the data stored in the circular buffer. The pixel data at the corresponding address is sent to the Epson controller.
Overlay_mode	A basic menu system is displayed. All input buttons are now directed to this function while in the menu system.

Table 10-2. Summary of Rabbit Programming Modules.

The final major programming module is the TCP/IP client that can run on any standard Linux PC. It is a command line program that takes two arguments: IP address of the Rabbit, and port nuMBer of the Rabbit server. The IP address argument can actually be a domain name that will automatically be resolved on execution. The program begins by scanning the local directory for picture files in the .ppm format. It has a built in picture pointer that starts off pointing to the first picture in the directory. It then tries to connect to the IP address and port coMBination passed via the command line. When connected, it constantly tries to read in command data from the Rabbit. Once the Rabbit server sends it a command, it executes the corresponding command and then returns to reading commands. The "nextpic" command simply changes the picture pointer to the next available picture file. The "prevpic" command changes the picture pointer to the previous available picture file. The "getpicname" command from the Rabbit causes the PC to respond by sending a "sendpicname" command back to the Rabbit with the name of the file pointed to by the picture pointer as the data field. When the PC receives the "recv" command, it must proceed to send the entire picture in order to the Rabbit. As it sends picture data, it checks to see if

another command has been received. Other commands can interrupt this command and take over execution. This is done so the user does not have to wait for an entire picture to display to cycle through to a different picture. In order to send a picture, the current picture file is broken down into pixels. Each pixel is converted from 24 bits per pixel to 15 bits per pixel because the Epson can only support a maximum of 15 bits per pixel without using a color table. First, the "sendnewpic" command is sent to the Rabbit that resets the Epson's address to the first address in the frame buffer and loads the first pixel color data. Every picture thereafter uses the "sendnextpixel" command to implicitly increment the address pointer as well as load in the corresponding pixel color data.

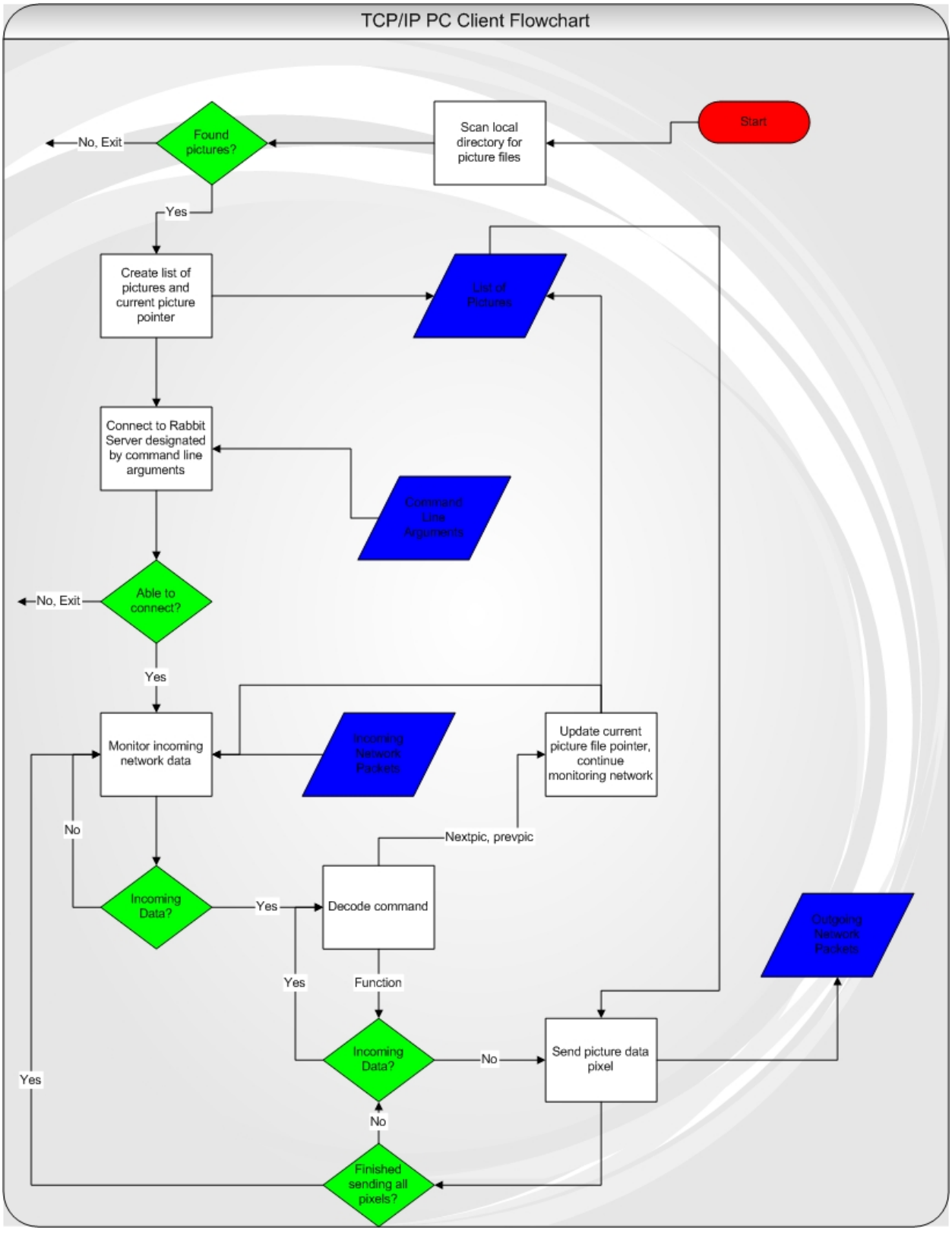


Figure 10-2. PC Client Flow Chart.

11.0 Version 2 Changes

After successfully completing our design project, there are several aspects we would change and improve if given a chance to revamp the final solution.

First, we would have tried to minimize the size of the board layout by adding two more layers: ground and power. This would have enabled us to route the Epson without so many problems. The Rabbit headers and the surface mount capacitors near the Epson could also have been routed much more easily. All of these components were surrounded by numerous 6 mil traces. We spent over 10 hours simply trying to route a power trace that was located on a surface mount capacitor in the midst of hundreds of 6 mil traces. We also could have packed all of the components more closely together in more strategic areas.

Next, we should have searched and asked for help until we found an actual 2 megabyte EDO DRAM chip. The final display on our project was slightly distorted due to the fact that we had a 4 megabyte DRAM chip, when the Epson datasheet explicitly stated that the maximum DRAM supported was 2 megabytes.

We should have combined all of the PLDs together into a larger variant so that we could have successfully implemented a 21 bit address counter. We could have just loaded the address in for the first pixel and then incremented the address on the PLD thereafter. This would have saved us the bandwidth required to send address data from the PC, eliminated the nuMBer of clock cycles required to clock pixels into the Epson, and sped up the process of picture display altogether.

We should have upgraded Dynamic C to the newest version at the beginning of software development. We discovered numerous bugs in Dynamic C 8.10 that wasted more than 20 hours of our time. Specifically, during the development of an alternate web server in case we could not get the Epson to work, countless hours were spent debugging code that was fine. We narrowed down the problem to Dynamic C's fwrite function. Whenever this function was called, it corrupted the costatement stack so that on return from a yield function it would return to a random spot. After upgrading to the newest version, our code worked as originally expected.

Finally, if we had more time we would have increased the functionality of the picture box. A wireless bridge could have easily been connected to the Rabbit's RJ-45 jack. We could have further added an on screen menu and a slideshow mode.

12.0 Summary and Conclusions

This entire project, from conception to realization, has unquestionably been one of the most memorable experiences of our college careers. It was quite an event to, after spending over 14 weeks working on a single project, see it actually work successfully. By utilizing a large portion of the knowledge and intuition gained from previous classes (and a large amount of knowledge obtained throughout the entire semester) we were able to implement something that could actually be manufactured in industry.

One of the most important things that we all learned early in the process was how to decipher datasheets and use that information to ensure compatibility between particular device components. This allowed us to have at least a small glimmer of hope that everything would "play nicely" together and ultimately produce a functioning device. Other technical skills acquired include new debugging techniques for (especially) hardware and also software. In terms of hardware, now more than ever we had to ensure that individual blocks of our design worked correctly before throwing it all together. It was incredibly reassuring to be able to hook up the Digital Logic Analyzer and see that the PLD's were, in fact, doing what was expected of them. The same general idea applied to software as well; many aspects of the programming were broken up into small pieces which, after ensuring they worked correctly, were combined with others to make the final version.

Ultimately this entire process has left us with what could easily be considered the final tools required to enter industry. Unlike the more theoretical courses, we started from scratch and came up with a fully functioning device to accomplish our goals. We relied solely on information publicly available from companies and used that to select components that would interface well and properly. We developed our own working schematic, created our own custom packaging, routed a printed circuit board, wrote our own software, and developed documentation for the end-user. In summary, we experienced the entire industry design process from start to finish. Through the late hours and brief confrontations, we accomplished something which quite frankly seemed incomprehensible, even impossible at the beginning of this semester.

13.0 References

[1] Rabbit 3010 Core Module

http://shay.ecn.purdue.edu/~477grp12/datasheets/rabbit3000_core_manual.pdf

[2] Atmel 22V10 PLD

http://shay.ecn.purdue.edu/~477grp12/datasheets/atmel_PLD.pdf

[3] Epson Graphics Controller

http://shay.ecn.purdue.edu/~477grp12/datasheets/epson_manual.pdf

[4] 4 MB EDO DRAM

http://shay.ecn.purdue.edu/~477grp12/datasheets/dram.pdf

[5] Epson 25.175 MHz Clock

http://shay.ecn.purdue.edu/~477grp12/datasheets/clock.pdf

[6] Texas Instruments Low-Dropout Voltage Regulators

http://shay.ecn.purdue.edu/~477grp12/datasheets/ldo.pdf

[7] Sharp IR Detector

http://shay.ecn.purdue.edu/~477grp12/datasheets/sharp_ir_detector_data.pdf

[8] Reynolds Electronics IR Decoder

http://shay.ecn.purdue.edu/~477grp12/datasheets/rentron_ir_decoder.pdf

[9] Cirrus Logic System-on-Chip with CRT/LCD Controller

http://shay.ecn.purdue.edu/~477grp12/datasheets/cirruslogic.pdf

[10] United States Patent and Trademark Office

http://www.uspto.gov

[11] United States Patent 6,037,989: *Still image transmitting device.*

[12] United States Patent 6,058,428: *Method and apparatus for transferring digital images on a network.*

[13] United States Patent 6,111,586: *Electronic photo album editing apparatus.*

[14] United States Patent 6,167,469: *Digital camera having display device for displaying graphical representation of user input and method for transporting the selected digital images thereof.*

[15] United States Patent 6,442,573: *Method and apparatus for distributing picture mail to a frame device community*

[16] CEIVA Logic, Inc.:

http://www.ceiva.com

*Note: References [11]-[15] found by performing patent search at [10]

[17] "Designing for Reliability, Maintainability and Safety – Parts 1, 2 and 3", Circuit Cellar, December 2000, January 2001, April 2001.

[18] MIL-HDBK-217F Reliability Prediction of Electronic Equipment

http://shay.ecn.purdue.edu/~dsml/ece477/Homework/Spring2004/Mil-Hdbk-217F.pdf

[19] Digi-Frame DF-1710

http://www.digi-frame.com/df1710.html

[20] Vosonic Multi-Media Viewer

http://www.vosonic.co.uk/mmv80.html

[21] Richard Stevens. Unix Network Programming – Networking APIs: Sockets and XTI. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[22] Dynamic C – TCP/IP User's Manual. Davis, CA: Z-World, Inc., 2002.

Appendix A: Individual Contributions

Contributions of Phillip Boone:

Initially I searched for a graphics controller, an EDO DRAM chip, and a JPEG decoder chip. I found a Cirrus Logic graphics controller that met all of our design constraints. I also looked for an evaluation board for the Epson chip Jeff found so that we could try and prototype our design. I sent out various emails to companies asking about products that I thought were actual JPEG decoder chips asking for samples. I could not find an actual chip that decoded JPEG image data, only ASIC software modules. I then found a Texas Instruments DM_270 media processor that included JPEG decoding and everything else our project needed. I asked for samples, but it turned out that this chip was not yet completed finished. Finally, I searched for an EDO chip for a few days on the internet. After I couldn't encounter one I brought several old EDO DRAM SIMM's that I had at home to see if we could use them.

I also helped solve a few hardware issues that we encountered. I discussed the glue logic necessary to latch in the 21-bit address into the Epson using only 16 Rabbit output pins with Jeff. We wanted a single clock to drive both the Epson and the Rabbit. I figured out where the clock was generated on the Rabbit core module so that we could route that to the Epson. We were later told to simply use an external clock. I also tried to figure out how to generate a clock on one of the Rabbit output pins. Finally, I helped with the layout homework. I routed part of the Epson controller, the PLD's, and a header.

I obtained the Rabbit 3000 microcontroller from Chuck. I printed up, read through partly, and had several Rabbit manuals bound. I brought the Rabbit evaluation board home and setup various test programs. I setup the cooperative multitasking environment on the Rabbit using costatements. I figured out the protocol necessary for communication between the Rabbit and PC. I wrote all of the software code for the Rabbit except for the functions that Jeff wrote to clock in color data into the Epson's frame buffer. This included code to debounce the switches, handle IR input, dump incoming Ethernet data into a circular buffer, and finally decode data stored in the circular buffer. Just in case the Epson did not work, I wrote a web server on the Rabbit that could dynamically change the contents of a file on the web page. This required implementing a file system on the Rabbit's SRAM, and dynamically changing the contents of a certain file with picture data using Dynamic C's file system functions. I had a working web

server that had a link to a JPG file. Once the connected PC sends a picture to the Rabbit, it was possible to see the changes on a remote computer by using any standard web browser. This proved that we had successfully received image data on the Rabbit.

I wrote all of the software for the PC client. I figured out how to setup a TCP/IP connection, send data, and receive data. I implemented the code that scans the local directory for picture files, and constructs a list of all files found. I wrote the code that decodes all of the data in the PPM picture file, scales the colors from 8 bits to 5 bits, and formats the color data so the Epson can use it. I setup the wait loop that detects incoming commands from the Rabbit server, and acts accordingly. I debugged the final version of the PC client software. I worked with the final version of the picture box, and tweaked the software so that it would stop displaying picture data after an absolute address. This was done to account for the fact that we were using a 4 megabyte EDO DRAM chip when the Epson required a 2 megabyte chip.

I helped construct the casing of our final picture box. I cut most of the Plexiglas pieces, glued them together, and helped make sure the box still worked in our casing.

I think that I put in my fair share for our project and I should receive the full grade that our project deserves. I directly helped satisfy every one of the four outcomes that we satisfied. I did not miss any group meetings or progress briefings.

Contributions of Bill Kreider:

Over the course of the semester, I performed several very important tasks that directly contributed to the success of the project. First, I researched and ordered the majority of the components that went into the design, excluding the Epson, DRAM, and Atmel PLDs. This required reading and downloading datasheets into the group account for further use. I designed the Infrared subsystem, both evaluating the performance in prototype and in the final PCB implementation. I assisted in testing the IR software and debugging the code after the initial implementation. After acquiring from Jeff the functionality of the Epson and its interactions with the PLDs, I completed the Schematic and Theory of Operation and maintained this throughout the semester as certain components changed within the project. I assisted with the layout by looking up the footprints for our components while Jeff entered them into the computer. The majority of the soldering was done during a week when I had two other exams, and thus I was unable to assist during this time. However, as problems arouse throughout the

semester, I would solder as needed. I did solder the DRAM chip and a few other small components. I assisted as needed during most of the Epson debugging, helping Jeff run his test code and interpret the Logic Analyzer responses. I was also part of the final stages of implementing Phil's software on the device, and I tested the board after a major component failure to ensure that no other component was in danger. I also completed the Patent Analysis homework, and worked with the group in selecting and assembling the final casing for the device. Throughout the semester, I also maintained sole responsibility for the team website. I conceived the initial design and created templates for each team member to use for updating their notebooks, if desired. I was responsible for organizing the presentations, and I updated all midterm slides for the final presentation. From the very beginning, I attempted to organize group meetings and delegate individual weekly tasks. I felt that in order to motivate the group, we all needed to be on the same page week after week. As the semester progressed, it became increasingly difficult to organize group meetings, and I spent time mediating arguments between members of the team. I feel the important aspects of the project were completed successfully and that the success criterion that was not fulfilled was miniscule compared to the others. As such, with the overall success of the project and the work that I put forth, I feel that I deserve an A in ECE 477.

Contributions of Egomaron Jegede:

My contributions to the project over the semester are as follows:

I worked closely with Phil in the development of software for the PC client and Rabbit in order to communicate and transmit data. I contributed ideas for the structure of the software and researched the development environment for the rabbit. I wrote a .ppm decoder to format and send information pixel by pixel to the rabbit. Jeff's ppm decoder was the version used in the PC client software module. I modified, tested and debugged several revisions of the software to communicate between the PC and the Rabbit. (Phil set-up the network and socket programming code). As a back-up plan to achieving the third outcome which was the ability to receive jpeg data via Ethernet, I worked with Phil on the creation of a web server to be run on the rabbit that could be accessed with a web client program to view the graphical data. This was proposed as a viable option in the event the EPSON could not be successfully interfaced with the rabbit. I helped modify sample code to customize and streamline its functionality as it originally included

an FTP server and spent a significant amount of time debugging, testing and modifying the web server code to completion.

I drafted the initial block diagram showing the major functional blocks of our design. I researched available rabbit microprocessors and development kits to determine part specifics, prices and suitability for our project. I created and updated the conceptual illustrations for the packaging of our design, contacted companies in order to obtain custom made casing and when this was not feasible, purchased the materials and oversaw the construction of our plexi-glass casing (done by all group members).

Regarding homework contributions I participated in all group exercises as we shared the work equally and completed my sections on time. I helped route the final few almost impossible traces on the PCB Layout. Individually I completed the Packaging Specifications and Design as well as the Reliability and Safety Analysis and FMECA worksheet. I compiled and edited the User Manual including an updated and annotated version of the product illustration. I reviewed, edited and contributed ideas to the individual homework completed by team members and participated fully in group demonstrations and presentations. I attended and contributed ideas and updates in the weekly progress briefings with course advisors through the course of the semester. I contributed significantly to achieving the defined success criteria for this senior design project and deserve an A in this course.

Contributions of Jeff Turkstra:

I spent an incredible amount of time this semester working to ensure that our project would be successful (and that I would graduate) upon completion of this semester. What follows is a pseudo-list of what I've done this semester.

For starters, I helped select just about every component used for the project. I also ordered quite a few of the components (Epson, DRAM, PLD's, etc). Moreover, I provided significant help to Bill during the schematic homework (I feel that at least half of the schematic was done by me). I also did the PCB layout with very little assistance from others. Twice. The first layout suffered from the infamous screwed up rabbit headers. I also did almost all of the board propagation once it arrived. Notable exceptions include the DRAM, the Epson chip (done by Chuck), a couple of pushbuttons, and a capacitor or two. I did all of the fly wiring.

In terms of the PLD's that were used in this project, I alone figured out how to use CUPL to program them and tested them thoroughly before soldering them onto the board. I also came up with solution to the Dataman checksum issue that I encountered.

PLD's aside, I took care of anything directly relating to the Epson. I wrote all of the code to provide the basic interfacing routines (writing a 16-bit value to any given address, making the screen entirely one color, etc). I also did all of the testing and debugging with respect to the Epson. This includes, but isn't limited to, connecting and using the Digital Logic Analyzer, generating test signals with the PLD's and verifying them with a DMM, and modifying the initial interfacing code to read back from the Epson's data registers.

I additionally provided the source code for a program that I wrote to decode a PPM file, which is the basis for Phil's PPM decoding code (mine was written in C++).

I also wrote the code responsible for driving the LED's as well as the initial code used to poll the IR and pushbuttons (Phil was responsible for the final code).

I installed and configured the NewsPro CGI script that Bill and I used to maintain our notebooks on the website.

I provided significant assistance with construction of the actual package that houses our device.

I sacrificed a video card for our VGA connector, and when that didn't appear to be working, a VGA port from my VGA switch box.

I was responsible for and did all of the video editing for our group video.

Of course, I was also the team leader. I think that I played a crucial role in this project's success…so much so that I believe my efforts alone resulted in the *initial* satisfaction of three of our established outcomes. With the ultimate success of our project, less one outcome which was quite frankly added at the last minute, I feel that I should receive an A in ECE 477.

Appendix B: Packaging

Figure B-1: Front View of *initial* DiPFI packaging concept

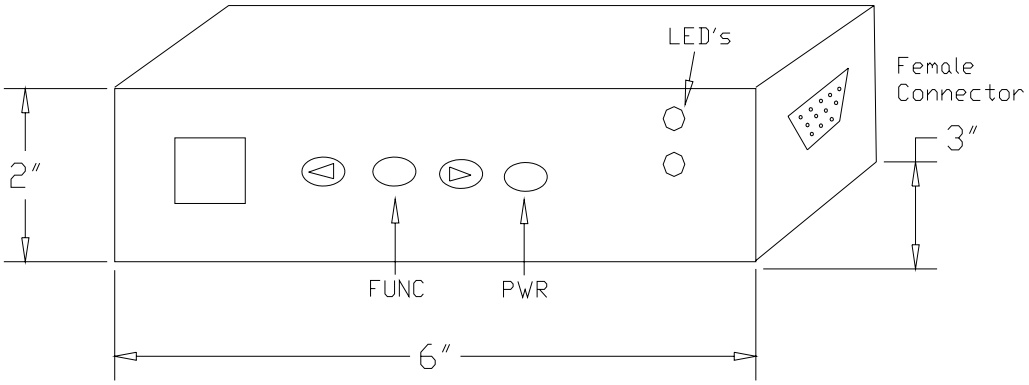
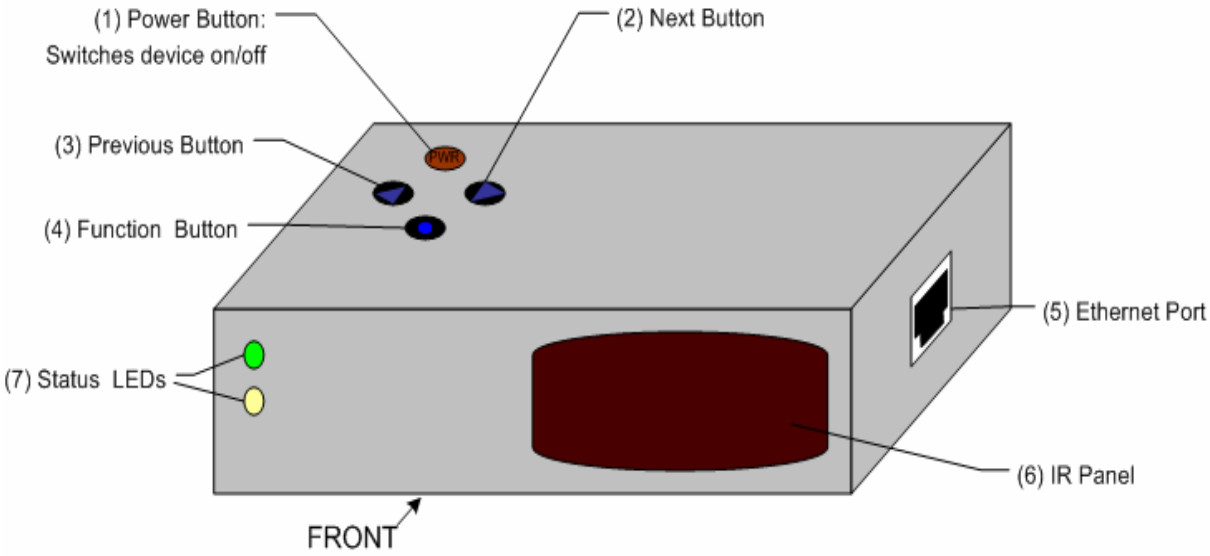


Figure B-2: Back View of *initial* DiPFI packaging concept

Figure B-3: Front and Top View of *final* DiPFI packaging



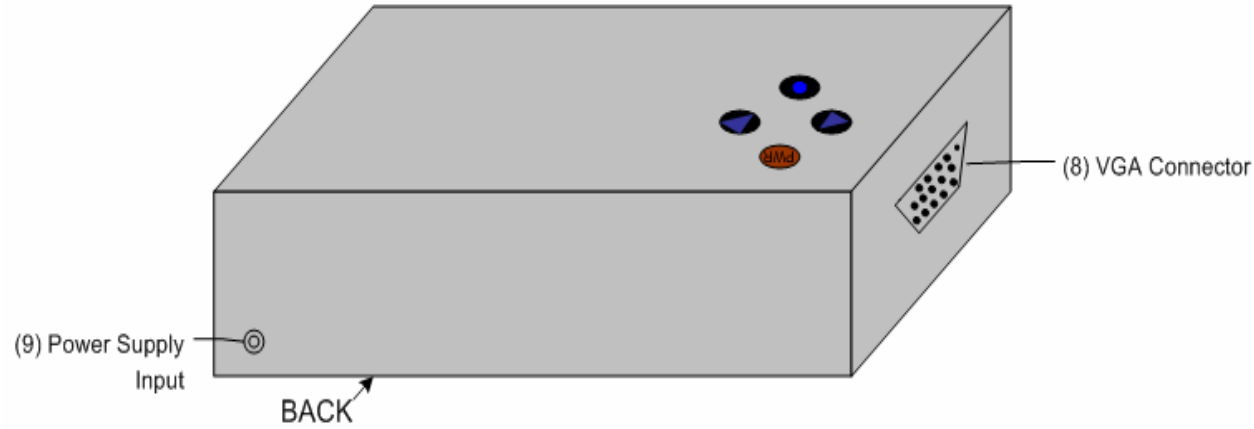


Figure B-4: Back View of *final* DiPFI packaging

Dimensions: 8.7 (l) x 7.7 (w) x 2.5 (h) (inches)
22 x 19.5 x 6.35 (centimeters)

Appendix C: Schematic

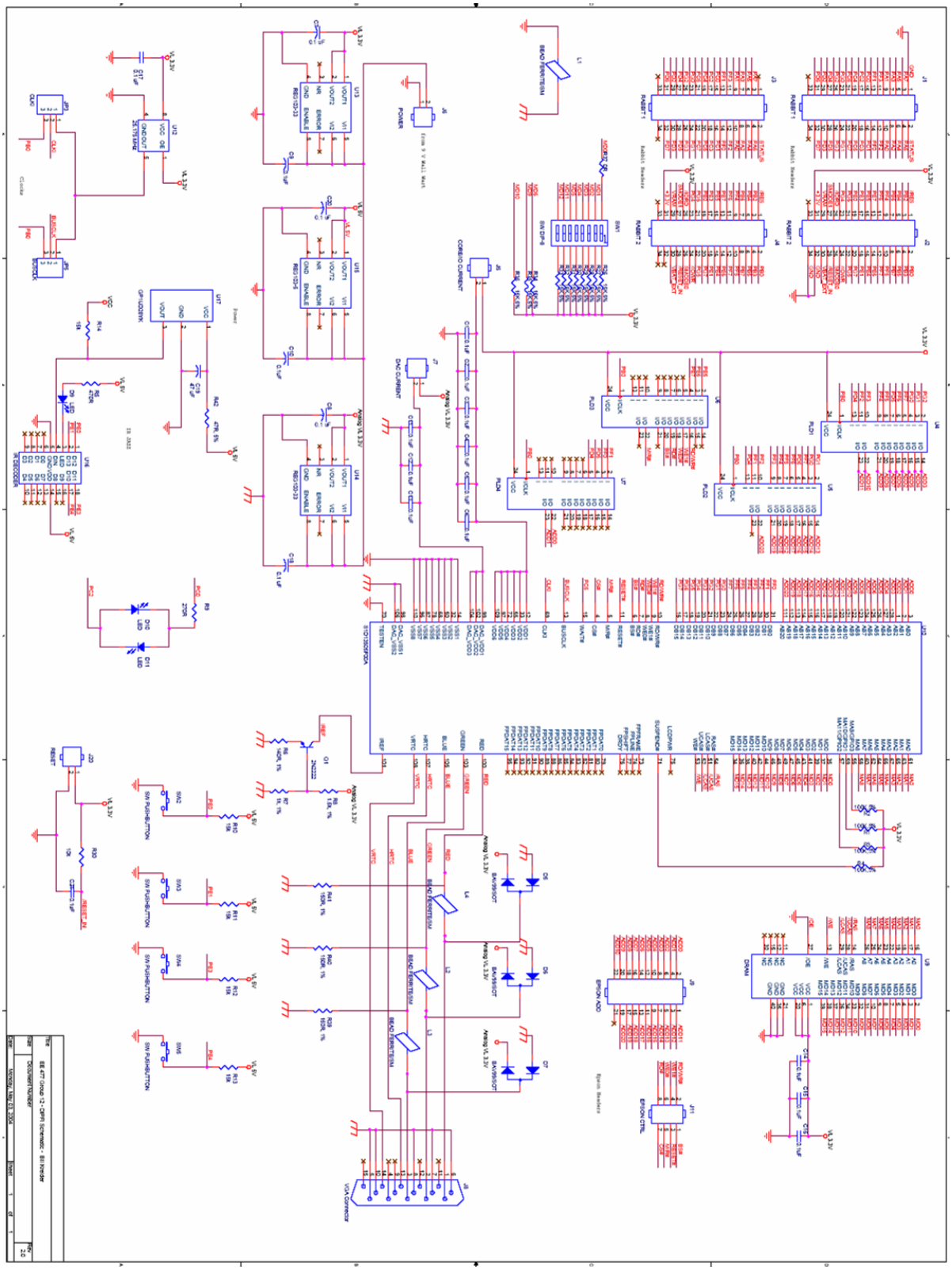


Figure C-1. OrCAD Schematic of DiPFI.

Appendix D: PCB Layout Top and Bottom Copper

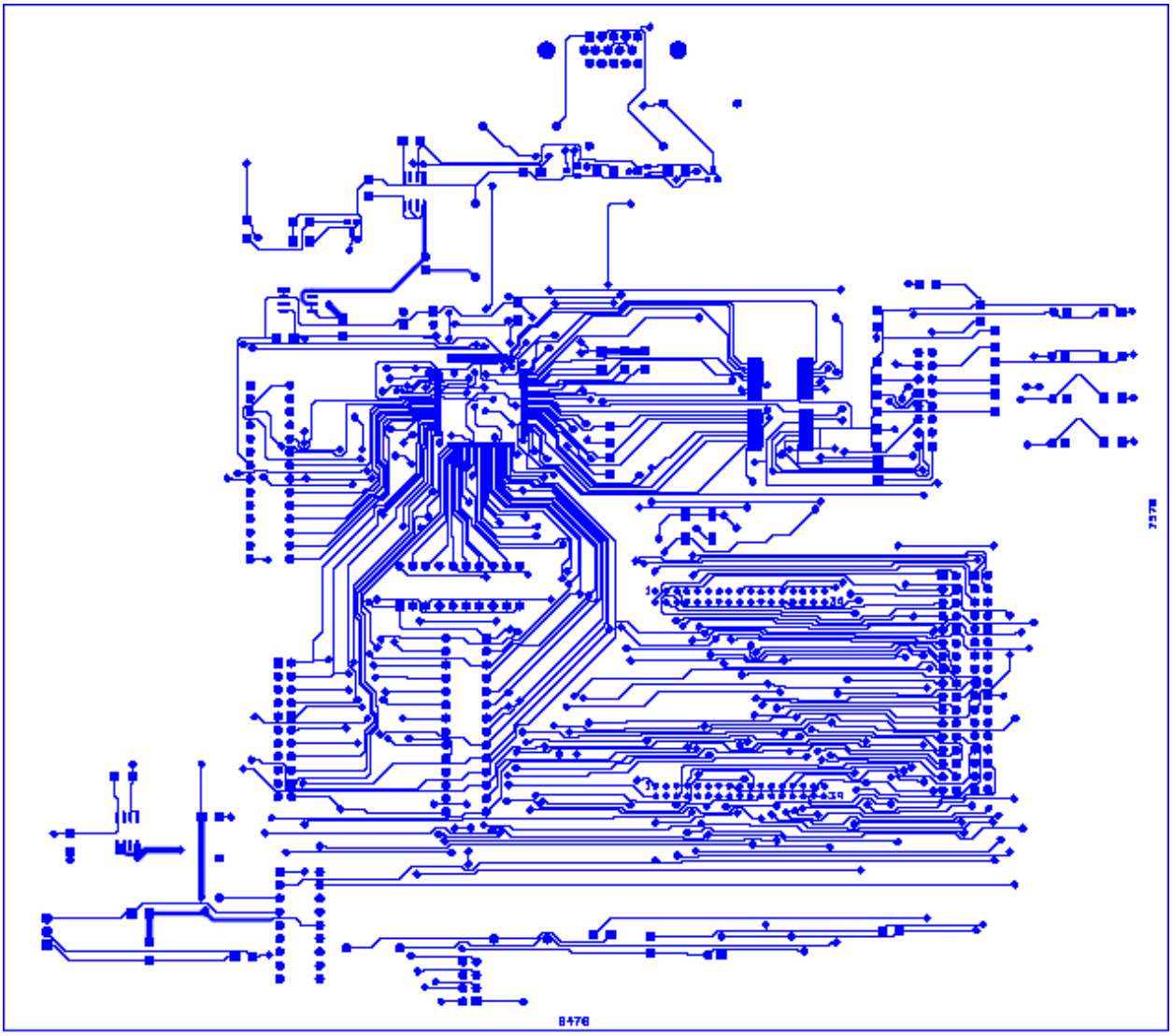


Figure D-1. PCB Layout Top Copper.

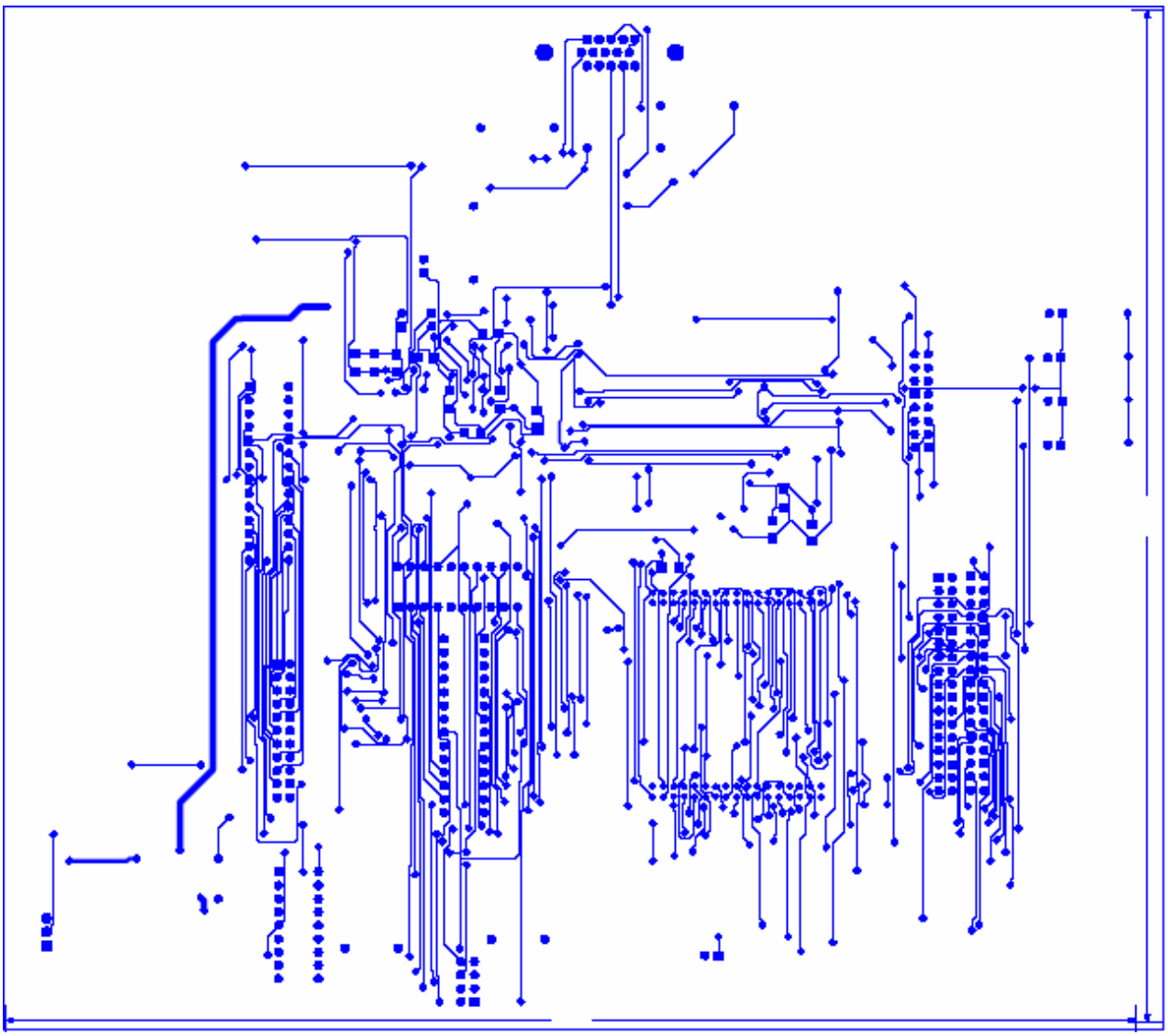


Figure D-2. PCB Layout Bottom Copper.

Appendix E: Parts List Spreadsheet

DiPFI Component List					
Vendor	**Part Description**	**Part Number**	**Unit Cost**	**Quantity**	**Total Cost**
Arrow Electronics, Inc.	Epson Graphics Controller	S1D13505F00A100	$14.40	4	$57.60
G-Link Technology	256Kx16 4MB EDO DRAM	GLT440L16	$3.00	3	$9.00
Digi-Key	Rabbit 3000 Core Module	316-1018-ND	$83.21	1	$83.21
Rabbit Semiconductor	Rabbit 3010 Core Module	101-0507	$59.00	2	$118.00
Digi-Key	Ferrite Beads	P9820BK-ND	$0.127	10	$1.27
Digi-Key	47 uF Caps	493-2086-1-ND	$0.178	5	$0.89
Digi-Key	Epson 25.175 MHz Clock	SE2838CT-ND	$4.38	2	$8.76
Digi-Key	0.1 uF Caps	493-2208-1-ND	$0.158	10	$1.58
Reynolds Electronics	Sony IR Decoder	IR-D14 IC	$8.000	1	$8.00
Arrow Electronics, Inc.	Sharp 40 kHz IR Receiver	GP1UD28YK	$0.750	3	$2.25
	Caps		Sampled	20	$0.00
	Resistors		Sampled	20	$0.00
Radio Shack	1500 mA 9V AC Adapter		$22.000	1	$22.00
Texas Instruments	500 mA LDO Volt. Regs.	REG 103-33/5	Sampled	25	$0.00
Texas Instruments	1 A LDO Volt. Regs.	TPS76750-33/5	Sampled	10	$0.00
Fairchild Semiconductor	High Conductance Ultra Fast Diode	BAV99	Sampled	6	$0.00
Fairchild Semiconductor	NPN General Purpose Amplifier	MMBT2222	Sampled	3	$0.00
	15 Pin Female VGA Connector		Sampled	2	$0.00
	Headers		Sampled	8	$0.00
	Pushbuttons		Sampled	6	$0.00
	LEDs		Sampled	4	$0.00
	Sony IR Remote		Sampled	1	$0.00
Digi-Key	Atmel 22v10 PLDs	ATF22LV10CQZ	$5.38	7	$37.66
	PCB		$33.00	1	$33.00
	Plexi-glass Sheet		$12.00	1	$12.00
	Poxy Glue		$6.00	1	$6.00
	RJ-45 Female to Female Connector		$5.00	1	$5.00
	Sand paper		$2.00	1	$2.00
				TOTAL	**$408.22**

Appendix F: Software Listing

```
/*****************************************************************************
    Custom TCP/IP Server Implemented on the Rabbit 3000 microcontroller
    that waits for incoming picture data, and forwards the data to the
    EPSON graphics controller.

                    By: Phillip Boone, Jeff Turkstra
*****************************************************************************/
#class auto

// Enable DHCP autoconfiguration on the network port
//#define TCPCONFIG 5

// Enable following lines to set STATIC IP
#define TCPCONFIG 1
#define _PRIMARY_STATIC_IP          "192.168.1.101"
#define _PRIMARY_NETMASK            "255.255.255.0"
#define MY_NAMESERVER               "192.168.1.1"
#define MY_GATEWAY                  "192.168.1.1"

#define TIMEZONE       -8

// Server listens on this port
#define PORT1 39
#define SOCK_BUF_SIZE 4096
#define MAX_BUFSIZE 128 // Blocksize of packets to read in
#define MAX_BUFFER 8191 // Temporary buffer of incoming packets
#define  CH_ESCAPE      27
#define SAMPLES 3

#memmap xmem
#use "dcrtcp.lib"                       //  Always pull in FS MkII.

// Temporary Address and Color data
typedef unsigned char S1D_INDEX;
typedef unsigned char S1D_VALUE;

typedef struct
{
   S1D_INDEX Index;
   S1D_VALUE Value;
} S1D_REGS;


// Initialiaztion data for the Epson's Registers
static S1D_REGS aS1DRegs[] =
{
   {0x1B,0x00},   // Miscellaneous Register
   {0x23,0x3B},   // Performance Enhancement Register 1
   {0x01,0x30},   // Memory Configuration Register - 0x20?
   {0x22,0x24},   // Performance Enhancement Register 0
   {0x02,0x26},   // Panel Type Register
   {0x03,0x00},   // MOD Rate Register
   {0x04,0x4F},   // Horizontal Display Width Register
```

```
    {0x05,0x13},   // Horizontal Non-Display Period Register
    {0x06,0x01},   // HRTC/FPLINE Start Position Register
    {0x07,0x0B},   // HRTC/FPLINE Pulse Width Register
    {0x08,0xDF},   // Vertical Display Height Register 0
    {0x09,0x01},   // Vertical Display Height Register 1
    {0x0A,0x2B},   // Vertical Non-Display Period Register
    {0x0B,0x09},   // VRTC/FPFRAME Start Position Register
    {0x0C,0x01},   // VRTC/FPFRAME Pulse Width Register
    {0x0E,0xFF},   // Screen 1 Line Compare Register 0
    {0x0F,0x03},   // Screen 1 Line Compare Register 1
    {0x10,0x00},   // Screen 1 Display Start Address Register 0
    {0x11,0x00},   // Screen 1 Display Start Address Register 1
    {0x12,0x00},   // Screen 1 Display Start Address Register 2
    {0x13,0x00},   // Screen 2 Display Start Address Register 0
    {0x14,0x00},   // Screen 2 Display Start Address Register 1
    {0x15,0x00},   // Screen 2 Display Start Address Register 2
    {0x16,0x80},   // Memory Address Offset Register 0
    {0x17,0x02},   // Memory Address Offset Register 1
    {0x18,0x00},   // Pixel Panning Register
    {0x19,0x00},   // Clock Configuration Register
    {0x1A,0x00},   // Power Save Configuration Register
    {0x1C,0x00},   // MD Configuration Readback Register 0
    {0x1E,0x00},   // General IO Pins Configuration Register 0
    {0x1F,0x00},   // General IO Pins Configuration Register 1
    {0x20,0x00},   // General IO Pins Control Register 0
    {0x21,0x00},   // General IO Pins Control Register 1
    {0x23,0x3B},   // Performance Enhancement Register 1
    {0x0D,0x12},   // Display Mode Register
};


static tcp_Socket sock1;
// IR Temporary Status
int PE0[2];
int PE1[2];
int PE3[2];
int PE4[2];

// Switch Debouncing Temp. variables
int PA0[SAMPLES + 1];
int PA1[SAMPLES + 1];
int PA2[SAMPLES + 1];
int PA3[SAMPLES + 1];

// Indices for the temp. arrays
int button;
int stable;
int ir;

// Temporary buffer to store data copies directly from the network socket
char buf1[MAX_BUFSIZE];
char indata[MAX_BUFFER];
int i_prod;
int i_cons;

// Address to clock into the EPSON
```

```
unsigned long total_size;
// 16 bit Color Value to clock in to the Epson
int pixeldata;
int activeled;

void led();

void clockit() {  // routine to clock the LED's
  BitWrPortI(PBDR, &PBDRShadow, 1, 0);
  BitWrPortI(PBDR, &PBDRShadow, 0, 0);
  return;
}

void setmode(int b0, int b1) { // configure the PLD's mode
  BitWrPortI(PDDR, &PDDRShadow, b0, 5);
  BitWrPortI(PDDR, &PDDRShadow, b1, 4);
  return;
}

// Load the Address into the PLDs
void loadaddy(long address) {
  setmode(0,1);
  // (Epson must have WORD-ALIGNED addresses!
  WrPortI(PFDR, &PFDRShadow, (address & 0xFE)); // lower 8 bits of address
  WrPortI(PGDR, &PGDRShadow, ((address & 0x7FF) >> 8)); // bits 8-10
  clockit();
  setmode(1,0);
  WrPortI(PFDR, &PFDRShadow, ((address & 0x07F000) >> 11)); // bits 12-18
  WrPortI(PGDR, &PGDRShadow, (((address & 0x7FFFFF) >> 19) +
    ((address & 0x800) >> 9))); // bits 19-20 & 11
  clockit();
  return;
}

// Put the Epson in the proper mode
void epsonmod(int mode) {
  /* 0 = read registers
     1 = write registers
     2 = read data
     3 = write data */
  if (mode == 0) {
    BitWrPortI(PEDR, &PEDRShadow, 0, 5);
    BitWrPortI(PEDR, &PEDRShadow, 0, 6);
    BitWrPortI(PEDR, &PEDRShadow, 0, 7);
  }
  else if (mode == 1) {
    BitWrPortI(PEDR, &PEDRShadow, 1, 5);
    BitWrPortI(PEDR, &PEDRShadow, 0, 6);
    BitWrPortI(PEDR, &PEDRShadow, 0, 7);
  }
  else if (mode == 2) {
    BitWrPortI(PEDR, &PEDRShadow, 0, 5);
    BitWrPortI(PEDR, &PEDRShadow, 1, 6);
    BitWrPortI(PEDR, &PEDRShadow, 0, 7);
  }
  else if (mode == 3) {
```

```
    BitWrPortI(PEDR, &PEDRShadow, 1, 5);
    BitWrPortI(PEDR, &PEDRShadow, 1, 6);
    BitWrPortI(PEDR, &PEDRShadow, 0, 7);
  }
  return;
}

// Write a byte "value" into the epson at "address"
void writeregbyte(unsigned long address, int value) {
  int i,j,orig;
  loadaddy(address);
  orig = readregcur();
  if ((address & 0x01) == 0) { // even address
    WrPortI(PFDR, &PFDRShadow, (value & 0xFF));          // load F
    WrPortI(PGDR, &PGDRShadow, (orig & 0xFF00) >> 8); // load G
  }
  else {
    WrPortI(PFDR, &PFDRShadow, (orig & 0xFF));
    WrPortI(PGDR, &PGDRShadow, (value & 0xFF));
  }
  epsonmod(1);
  BitWrPortI(PBDR, &PBDRShadow, 0, 3); // enable ChipSelect (activelow)
  while (BitRdPortI(PCDR, 5) == 0) ;   // busy wait until epson done
  BitWrPortI(PBDR, &PBDRShadow, 1, 3); // disable ChipSelect
  return;
}

int readregcur() {    // read a word from the address currently latched in PLD's
  int j, v1, v2;
  WrPortI(PFDDR, &PFDDRShadow, 0x00); // set ports f,g to input
  WrPortI(PGDDR, &PGDDRShadow, 0x00);
  epsonmod(0);
  BitWrPortI(PBDR, &PBDRShadow, 0, 3);  // enable chipselect
  v1 = RdPortI (PFDR);
  v2 = RdPortI (PGDR);
  BitWrPortI(PBDR, &PBDRShadow, 1, 3); // shutdown epson
  epsonmod(1); // well, anything that's input to avoid bus fighting
  WrPortI ( PFDDR, &PFDDRShadow, 0xFF);  // Ports F & G are output only
  WrPortI ( PGDDR, &PGDDRShadow, 0xFF);
  v1 = v1 & 0xFF; // bit masking to build the correct value
  v2 = v2 << 8;
  v1 = v1 + v2;
  return v1;
}

void readreg(unsigned long address) {
  int j, v1, v2;
  loadaddy(address);
  WrPortI ( PFDDR, &PFDDRShadow, 0x00); // set ports F & G to input
  WrPortI ( PGDDR, &PGDDRShadow, 0x00);
  epsonmod(0); // epson = out
  BitWrPortI(PBDR, &PBDRShadow, 0, 3);  // enable chipselect
  v1 = RdPortI(PFDR);
  v2 = RdPortI(PGDR);
  BitWrPortI(PBDR, &PBDRShadow, 1, 3); // shutdown epson
  printf("0x%x: ", (address & 0x1FFFFE)); // output data to screen
```

```
  printf("0x%x\n", v1);
  printf("0x%x: ", (address | 0x01));
  printf("0x%x\n", v2);
  epsonmod(1); // well, anything that's input to avoid bus fighting
  WrPortI ( PFDDR, &PFDDRShadow, 0xFF);  // Ports F & G are output only
  WrPortI ( PGDDR, &PGDDRShadow, 0xFF);
  return;
}

// Write a 16 bit color "value" into the Epsons frame buffer at "address"
void writedat(unsigned long address, int value) {
  // this routine has no protection against non-word aligned
  // addresses. It assumes that calls to it always provide a
  // word-aligned (even-numbered) address, and a 16-bit value
  // lower 8-bits correspond to data written to address
  // upper 8 correspond to data written to address+1
  loadaddy(address);
  WrPortI(PFDR, &PFDRShadow, (value & 0xFF));          // load F
  WrPortI(PGDR, &PGDRShadow, ((value & 0xFF00) >> 8));  // load G
  epsonmod(3);
  BitWrPortI(PBDR, &PBDRShadow, 0, 3); // enable ChipSelect (activelow)
  while (BitRdPortI(PCDR, 5) == 0) ;   // busy wait until epson done
  BitWrPortI(PBDR, &PBDRShadow, 1, 3); // disable ChipSelect
  return;
}

// Read a 16 bit color value from the Epsons Frame buffer
void readdat(unsigned long address) {
  // read 16-bits from DRAM space
  int j, v1, v2;
  loadaddy(address);
  WrPortI ( PFDDR, &PFDDRShadow, 0x00);  // Ports f,g input
  WrPortI ( PGDDR, &PGDDRShadow, 0x00);
  epsonmod(2); // epson = out
  BitWrPortI(PBDR, &PBDRShadow, 0, 3);  // enable chipselect
  v1 = RdPortI(PFDR);
  v2 = RdPortI(PGDR);
  BitWrPortI(PBDR, &PBDRShadow, 1, 3); // shutdown epson
  printf("0x%x: ", (address & 0x1FFFFE));
  printf("0x%x\n", v1);
  printf("0x%x: ", (address | 0x01));
  printf("0x%x\n", v2);
  epsonmod(1); // well, anything that's input to avoid bus fighting
  WrPortI ( PFDDR, &PFDDRShadow, 0xFF);  // Ports F & G are output only
  WrPortI ( PGDDR, &PGDDRShadow, 0xFF);
  return;
}

// This module just takes the data stored in the buffer indata
// and if it has an entire pixel packet, it clocks the pixel
// into the frame buffer of the Epson
cofunc int decodePCdata(tcp_Socket *s, int port)
{
  auto int i;
  char integer1[4];
```

```
  if( sock_established(s))
  {
   i = 0;
   while( i < 4 )
   {
      while( i_cons == i_prod ) yield;
      integer1[i] = indata[i_cons];
      i++;
      i_cons = (i_cons + 1) % MAX_BUFFER;
   }

   total_size = *( (unsigned long *) integer1);
   //printf( "Address: %lu", total_size );

   i = 0;
   while( i < 2 )
   {
      while( i_cons == i_prod ) yield;
      integer1[i] = indata[i_cons];
      i++;
      i_cons = (i_cons + 1) % MAX_BUFFER;
   }

   pixeldata = *( (int *) integer1);
   //printf( " Pixeldata: %x\n", pixeldata );

   led();
   writedat(total_size, pixeldata);
  }

  return 1;
}

// This modules listens to the ethernet port until there is a
// connection.  Once connected, it repeatedly drops all network
// data into the circular buffer indata.
cofunc int incoming_tcp(tcp_Socket *s, int port, char *buf)
{
  auto int length, space_available, ready, established, i;

  tcp_listen(s, port, 0, 0, NULL, 0);

  while( (-1 == sock_bytesready(s)) && (0 == sock_established(s)))
  {
    if(tcp_tick(s))
      yield;
    else
      abort;
  }

  while(sock_established(s))
  {
    space_available = sock_tbleft(s);

    if( space_available > (MAX_BUFSIZE-1))
      space_available = (MAX_BUFSIZE-1);
```

```
    length = sock_fastread(s, buf, space_available);

    if(length > 0)
    {
      // PACKET DATA NOW STORED IN buf
      // BEGIN PROCESSING HERE

      // Place data in circular buffer
      for( i = 0; i < length; i++ )
      {
        // Wait until buffer has enough space
        while( i_cons == (i_prod + 1) % MAX_BUFFER )
          yield;

        //printf( "Producing %c i_cons: %d i_prod: %d\n", buf[i], i_cons, i_prod );
        indata[i_prod] = buf[i];
        i_prod = (i_prod + 1) % MAX_BUFFER;
      }
    }
    yield;
  }
  sock_close(s);
  return 1;
}

// This module checks for changes in the bits on input ports PE0, PE1,
// PE3, and PE4.  If there is a change(an IR button was pressed), then
// send the corresponding command to the PC via network.
cofunc int monitor_ir( tcp_Socket *s, int port )
{
  auto int len, prev;

  prev = ir;

  // sample IR input polling
  ir = (ir + 1) % 2;

  PE0[ir] = BitRdPortI(PEDR, 0);
  PE1[ir] = BitRdPortI(PEDR, 1);
  PE3[ir] = BitRdPortI(PEDR, 3);
  PE4[ir] = BitRdPortI(PEDR, 4);

  if( (PE0[ir] ^ PE0[prev]) == 1 )
  {
    printf("IR PE0 pressed...\n");

    if( sock_established(s) )
    {
      // NOT USED
      //sock_fastwrite( s, "Recv\n", 5);
    }
  }

  if( (PE1[ir] ^ PE1[prev]) == 1 )
  {
```

```
    printf("IR PE1 pressed...\n");
    if( sock_established(s) )
      sock_fastwrite( s, "Function\n", 9);
  }

  if( (PE3[ir] ^ PE3[prev]) == 1 )
  {
    printf("IR PE3 pressed...\n");
    if( sock_established(s) )
      sock_fastwrite( s, "Previous Picture\n", 17);
  }

  if( (PE4[ir] ^ PE4[prev]) == 1 )
  {
    printf("IR PE4 pressed...\n");
    if( sock_established(s) )
      sock_fastwrite( s, "Next Picture\n", 13);
  }

  return 1;
}

// This module debounces the switches attached to ports PA2 through PA3.
// If a switch is pressed, then it sends the corresponding command to
// the PC via network.
cofunc int monitor_buttons( tcp_Socket *s, int port )
{
  auto int prev, prev1;

  while( DelayMs(50) == 0 )
    yield;

  prev = (button + 1) % SAMPLES;
  prev1 = (button + 1) %SAMPLES;

  PA0[button] = BitRdPortI(PADR, 0);
  PA1[button] = BitRdPortI(PADR, 1);
  PA2[button] = BitRdPortI(PADR, 2);
  PA3[button] = BitRdPortI(PADR, 3);

  if( (PA0[button] == PA0[prev]) && (PA0[prev] == PA0[prev1]) )
  {
    if( PA0[SAMPLES] != PA0[button] )
    {
      PA0[SAMPLES] = PA0[button];
      if( PA0[SAMPLES] == 1 )
      {
        printf("PA0 Pressed\n");
        if( sock_established(s) )
        {
          // NOT USED
          //sock_fastwrite( s, "Recv\n", 5);
        }
      }
    }
  }
```

```
  if( (PA1[button] == PA1[prev]) && (PA1[prev] == PA1[prev1]) )
  {
   if( PA1[SAMPLES] != PA1[button] )
   {
    PA1[SAMPLES] = PA1[button];
    if( PA1[SAMPLES] == 1 )
    {
     printf("PA1 Pressed\n");
     if( sock_established(s) )
      sock_fastwrite( s, "Function\n", 9);
    }
   }
  }

  if( (PA2[button] == PA2[prev]) && (PA2[prev] == PA2[prev1]) )
  {
   if( PA2[SAMPLES] != PA2[button] )
   {
    PA2[SAMPLES] = PA2[button];
    if( PA2[SAMPLES] == 1 )
    {
     printf("PA2 Pressed\n");
     if( sock_established(s) )
      sock_fastwrite( s, "Previous Picture\n", 17);
    }
   }
  }

  if( (PA3[button] == PA3[prev]) && (PA3[prev] == PA3[prev1]) )
  {
   if( PA3[SAMPLES] != PA3[button] )
   {
    PA3[SAMPLES] = PA3[button];
    if( PA3[SAMPLES] == 1 )
    {
     printf("PA3 Pressed\n");
     if( sock_established(s) )
      sock_fastwrite( s, "Next Picture\n", 13);
    }
   }
  }

  // sample button input polling counter
  button = (button + 1) % SAMPLES;

  return 1;
}

void led() {
  // utility to toggle the output led's (helps tell where we are in code
  // execution).
  if (activeled == 1) {
    BitWrPortI ( PCDR, &PCDRShadow, 1, 0);
    BitWrPortI ( PCDR, &PCDRShadow, 0, 2);
    activeled = 0;
```

```
  }
  else {
    BitWrPortI ( PCDR, &PCDRShadow, 0, 0);
    BitWrPortI ( PCDR, &PCDRShadow, 1, 2);
    activeled = 1;
  }
  return;
}

// Perform initialization, and enter into Round Robin process loop
void main()
{
  int       index, ftpUserID;
  int       rc;
  int i,j,k;
  unsigned char cval;
  unsigned long pMem, pTmp;

  ir = 0; // Counter for IR sampling
  button = 0; // Counter for button sampling
  i_prod = i_cons = 0; // Indexes of buffered data

  // init rabbit pins
  cval = RdPortI(PEDDR);
  WrPortI ( PEDDR, &PEDDRShadow, (cval | 0xE0) & 0xF0 ); // init Port E pins (upper 3 output)
  cval = RdPortI(PBDDR);
  WrPortI ( PBDDR, &PBDDRShadow, cval | 0x0D ); // use pins 0,2,3
  cval = RdPortI(PDDDR);
  WrPortI ( PDDDR, &PDDDRShadow, cval | 0x30 ); // mode select (only use bits 4&5)
  WrPortI ( SPCR, &SPCRShadow, 0x80);
  WrPortI ( PFDDR, &PFDDRShadow, 0xFF);  // Ports F & G are output only
  WrPortI ( PGDDR, &PGDDRShadow, 0xFF); // asdf

  BitWrPortI(PBDR, &PBDRShadow, 1, 3); // disable ChipSelect
  BitWrPortI(PBDR, &PBDRShadow, 0, 2); // reset Epson
  BitWrPortI(PBDR, &PBDRShadow, 1, 2);
  activeled = 0;
  led();

  // begin epson initialization sequence
  for (i = 0; i <35; i++) { // cycle through epson config values & load
    led();
    writeregbyte(aS1DRegs[i].Index,aS1DRegs[i].Value);
  }
  for (i = 0; i <36; i += 2) {
    led();
    readreg(i);
  }

  sock_init();
  printf( "setup complete.\n" );

   while (1)
   {
     costate
     {
```

```
      //printf("Incoming TCP\n");
      wfd incoming_tcp(&sock1, PORT1, buf1);
    }

    costate
    {
      //printf("Monitor IR\n");
      wfd monitor_ir(&sock1, PORT1);
    }

    costate
    {
      //printf("Monitor Buttons\n");
      wfd monitor_buttons(&sock1, PORT1);
    }

    costate
    {
      //printf("Decode PC Data\n");
      wfd decodePCdata(&sock1, PORT1);
    }

    costate
    {
      //printf("Tick\n");
      tcp_tick(NULL);
    }
  }
}



/*************************************************************************
     Custom TCP/IP Client that can compile on any Linux machine
     that scans the local directory for .ppm picture files, and
     waits for control commands from the Rabbit so that it can
     send picture data packets.

                        By: Phillip Boone
*************************************************************************/
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <errno.h>
#include <poll.h>
#define MAXPACKET 1024

#define FALSE 0
```

```
#define TRUE !FALSE

/* prototype std lib functions */
extern  int alphasort();

/* variable to store current path */
char pathname[MAXPATHLEN];

/* Wrapper function that connects to a TCP server with
   IPv4 address IP_ADDR.  If a domain name is given the
   name is automatically resolved to an IPv4 address. The
   file descriptor once connected is returned otherwise
   -1 is returned on error */
int Connect( char *IP_ADDR, int port_num )
{
  struct sockaddr_in ipaddr1;
  struct hostent *resolved;
  struct in_addr *temp1;
  int sockfd, error1;
  char IP1[INET_ADDRSTRLEN];
  char **res;

  bzero( &ipaddr1, sizeof(ipaddr1) );

  ipaddr1.sin_family = AF_INET;
  ipaddr1.sin_port = htons(port_num);

  if( inet_pton(AF_INET, IP_ADDR, &ipaddr1.sin_addr ) <= 0 )
  {
    if( (resolved = gethostbyname( IP_ADDR )) != NULL )
    {
      res = resolved->h_addr_list;

      if( inet_ntop( resolved->h_addrtype, *res, IP1, sizeof(IP1) ) == NULL )
        return -1;

      temp1 = (struct in_addr *)*res;

      ipaddr1.sin_addr.s_addr = temp1->s_addr;
    }
    else
    {
      return -1;
    }
  }

  if( (sockfd = socket( AF_INET, SOCK_STREAM, 0 ) ) == -1 )
  {
    return -1;
  }

  inet_ntop(AF_INET, &ipaddr1.sin_addr, IP1, sizeof(IP1) );
  if( IP1 == NULL )
  {
    return -1;
  }
```

```
  if( ( error1 = connect( sockfd, (struct sockaddr *)&ipaddr1, sizeof(ipaddr1) ) ) != 0 )
  {
    fprintf(stderr,"Unable to connect to %s\n", IP_ADDR  );
    return -1;
  }

  return sockfd;
}

/* Reads S_Buff nuMBer of bytes from socket Fd and stores
   them in Buff. */
int Read( int Fd, void *Buff, int S_Buff )
{
  int byte_num, byte_read;
  char *ptr;

  byte_num = S_Buff;
  ptr = Buff;

  while( byte_num > 0 )
  {
    if( (byte_read = read( Fd, ptr, byte_num ) ) < 0 )
    {
      if( errno == EINTR )
        byte_read = 0;
      else
      {
        fprintf( stderr, "Error Reading socket\n" );
        return S_Buff - byte_num;
      }
    }
    else if( byte_read == 0 ) break;

    byte_num -= byte_read;
    ptr += byte_read;
  }

  return S_Buff - byte_num;
}

/* Writes S_Buff nuMBer of bytes to socket Fd
   from buffer Buff. */
int Write( int Fd, void *Buff, int S_Buff )
{
  int byte_num, byte_wrote;
  char *ptr;

  byte_num = S_Buff;
  ptr = Buff;

  while( byte_num > 0 )
  {
    if( (byte_wrote = write( Fd, ptr, byte_num ) ) < 0 )
    {
      if( errno == EINTR )
```

```
      byte_wrote = 0;
     else
     {
      fprintf( stderr, "Error Reading socket\n" );
      return S_Buff - byte_num;
     }
    }
    else if( byte_wrote == 0 ) break;

    byte_num -= byte_wrote;
    ptr += byte_wrote;
  }

  return S_Buff - byte_num;
}

// Takes in command line arguments IP Address and Port, connects via
// Ethernet, and waits for command data from the Storage Box.
int main( int argc, char *argv[] )
{
  int sockfd, count, i, j, k, pic_sent, pdes[2];
  short Dlen;
  FILE *file1;
  char Buff1[MAXPACKET], temp, command[25], filename[256];
  struct direct **files;
  int file_select();
  struct pollfd Monitor;

  // .ppm Picture Information
  int width;
  int height;
  int quantLevels;
  char asdf[100];
  unsigned char pix1,pix2,pix3;

  // Final Processed pixel data for the Epson on the Storage Box
  unsigned int address;
  short value;

  if( argc != 3 )
  {
   fprintf(stderr, "Usage: %s <IP Address> <Port>\n", argv[0] );
   exit(1);
  }

  printf("Connecting to %s\n", argv[1] );

  printf("Port NuMBer is: %i\n", atoi(argv[2]) );

  sockfd = Connect( argv[1], atoi(argv[2]) );

  if( sockfd == -1 )
  {
   printf("Unable to connect\n");
   exit(1);
  }
```

```
if(getwd(pathname) == NULL )
{
  printf("Error getting path\n");
  exit(1);
}
printf("Current Working Directory = %s\n",pathname);

// NuMBer of .ppm files found
count = scandir(pathname, &files, file_select, alphasort);

/* If no files found, make a non-selectable menu item */
if (count <= 0)
{
  printf("No files in this directory\n");
  exit(0);
}

printf("NuMBer of .ppm files = %d\n",count);

for (i=1;i<count+1;++i)
 { printf("%s   ",files[i-1]->d_name);
   if ( (i % 4) == 0) printf("\n");
 }

printf("\n"); /* flush buffer */

// CONNECTED, START CODE

// Monitor socket for incoming data
Monitor.fd = sockfd;
Monitor.events = POLLIN;

// Open a communication pipe between parent/child
pipe(pdes);

i = 0;
j = 0;

if( fork() != 0 )
{
  close(pdes[0]); // Use pdes[1] to write to child

  while( Read( sockfd, &command[i], 1 ) == 1 )
  {
    if( command[i] == '\n' )
    {
      command[i+1] = '\0';
                write(pdes[1], command, i+1);
                i = 0;
    }
    else
      i++;
  }
}
else
```

```
{
 close(pdes[1]); // Use pdes[0] to read from parent

 i = 0;
 pic_sent = 0;

 while( 1 )
 {
        // Busy wait
        while( !read(pdes[0], &command[i], 1) );

        printf( "%c", command[i] );

        if( command[i] == '\n')
  {
         printf("\n");
   command[i+1] = '\0';
         i = 0;

   if( strcmp(command, "Next Picture\n") == 0 )
   {
    j = (j + 1) % count;
    printf("#### Selecting File %s ####\n", files[j]->d_name );
        }

   if( strcmp(command, "Previous Picture\n") == 0 )
   {
    j--;
    if( j == -1 ) j = count - 1;
    printf("#### Selecting File %s ####\n", files[j]->d_name );
        }

        if( strcmp(command, "Recv\n") == 0 )
   {
         if( read(pdes[0], &pic_sent, 4) != 4)
         {
     printf("ERROR: Unable to Receive length\n");
          exit(1);
         }

    // GET NULL TERMINATED FILE NAME
        k = 0;
    if( pic_sent > 0 )
    {
     do
         {
          while( !read(pdes[0], &command[0], 1) );

          filename[k] = command[0];
                k++;
                pic_sent--;
         } while( command[0] != '\0');
        }

        file1 = NULL;
```

```
            if( pic_sent > 0 )
            {
   printf("######## File %s of size %i ########\n", filename, pic_sent);
   printf("Storing new file to %s...\n", strcat(filename, ".rab") );
   file1 = fopen(filename, "w");
            }

            while( pic_sent )
            {
              while( !read(pdes[0], &command[0], 1) );
              //printf("%c", command[0] );
              fputc(command[0], file1);
              pic_sent--;
            }

            if(file1 != NULL) fclose(file1);

            pic_sent = 0;

} // END OF IF RECV

          if( strcmp(command, "Function\n") == 0 )
{
 file1 = fopen(files[j]->d_name, "r");

 printf("#### Sending File %s to the Rabbit ####\n", files[j]->d_name );

 while( fscanf(file1, "#%[^\n]\n", asdf) == 1 );

 fscanf(file1, "%[^\n]\n", asdf);
 printf( "%s\n", asdf );

 fscanf(file1, "%d %d\n", &width, &height );
 printf( "Widght: %i\n", width );
 printf( "Height: %i\n", height );

 fscanf(file1, "%d\n", &quantLevels );
 printf( "Quant: %i\n", quantLevels );

 address = 0;

 while( !feof( file1 ) )
 {
  pix1 = fgetc(file1);
  pix2 = fgetc(file1);
  pix3 = fgetc(file1);

  // Scale the data from 8 bits to 5 bits
  pix1 = ((pix1*32)/256);
  pix2 = ((pix2*32)/256);
  pix3 = ((pix3*32)/256);

  // Pack the data into one 16 bit color data value
  value = pix3 | (pix2 << 5) | (pix1 << 10);

  if( Write( sockfd, &address, 4 ) != 4 )
```

```
          {
           fprintf(stderr, "Unable to write picture frame packet\n" );
           return 0;
          }

          if( Write( sockfd, &value, 2 ) != 2 )
          {
           fprintf(stderr, "Unable to write picture frame packet\n" );
           return 0;
          }

          if( poll(&Monitor, 1, 0) < 0 )
            printf( "Unable to Poll Incoming Ethernet Data\n");

          if( Monitor.revents != 0 )
            printf("%i\n", Monitor.revents);

          //printf("Scaled %u: R: %x G: %x B: %x Value: %x\n", address, pix1, pix2, pix3, value);

          // Account for DRAM overlap and the Fact that we have the wrong
          // Size chip.  This is not necessary if we had the right DRAM.
          if( address >= 525500 ) break;

          address++;
          address++;
         } // END WHILE FEOF

         fclose(file1);
         printf( "\n\n");
       } // END IF FUNCTION
     } // END IF COMMAND RECEIVED
            else
       i++;
    } // END BIG INFINITE WHILE
  }
  close(sockfd);
  return 0;
}
/* Only select files that end in .ppm, and are not  . or .. */
int file_select(struct direct *entry)
{
  char *ptr;
  if( (strcmp(entry->d_name, ".") == 0) ||
      (strcmp(entry->d_name, "..") == 0) )
    return (FALSE);

  ptr = rindex(entry->d_name, '.');

  if ( (ptr != NULL) && (strcmp(ptr, ".ppm") == 0) )
    return (TRUE);
  else
    return(FALSE);


  return (TRUE);
}
```

Appendix G: User Manual

DiPFI

Digital **P**icture **F**rame **I**nterface

"Picture your world."

User Manual

Contents

Congratulations!!

You have just purchased the latest in digital picture technology. With the Digital Picture Frame Interface, you have chosen to revivify your old VGA devices, and turn them into highly portable digital image viewers. The device is simple to use, requiring only a standard Ethernet cable[(1)] and a standard 9 Volt power supply for full functionality. Simply attach DiPFI to a VGA monitor, supply it with an Ethernet connection and power, and you are ready to view any digital picture[(2)] stored on your configurable home personal computer.

The device has onboard pushbuttons and status LEDs which allow for a simple and efficient means for communicating with the display. In addition, DiPFI offers infrared remote capabilities, which allows you the benefit of controlling the device from across the room using the volume and channel buttons on any remote using the Sony IR Remote Protocol!!

You may ask, "How does it work?" Just set up your VGA device on a shelf for prominent display, plug in power and Ethernet to your new picture frame interface and attach DiPFI to the VGA device. Next, connect to DiPFI using your personal computer and DiPFI is now ready to let you picture your world for all to see. So sit back, and enjoy your memories.

[(1)]Not included [(2)]Works on 640 x 480 JPEG images

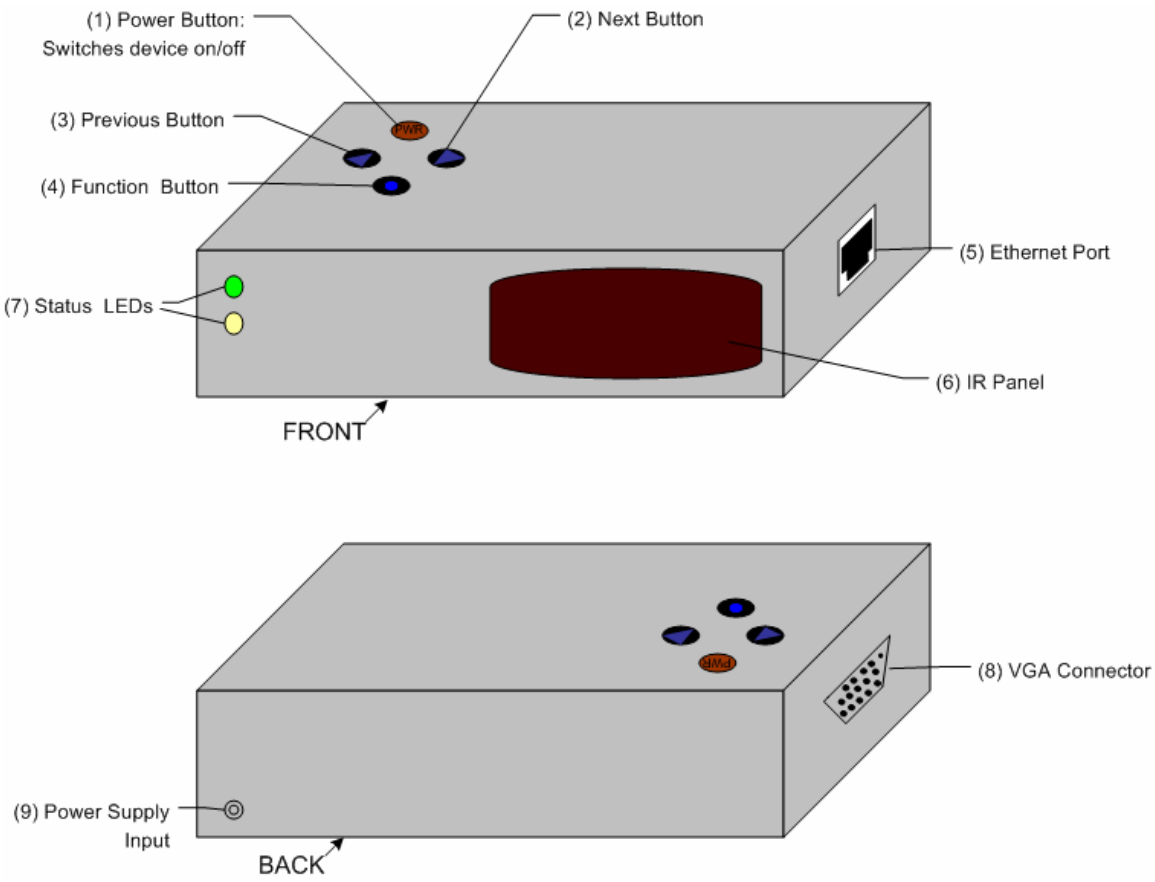


Figure 1. DiPFI Front and Back Views

Dimensions: 8.7 (l) x 7.7(w) x 2.5(h) (inches)
22 x 19.5 x 6.35 (centimeters)

Component	Function
(1) Power Button	Used to switch the device on and off.
(2) Next Button	Sends the next picture from the PC to the display.
(3) Previous Button	Sends the previous picture from the PC to the display.
(4) Function Button	Puts the device into and out of slideshow mode.
(5) Ethernet Port	RJ-45 port for connecting the Ethernet cable from the PC.
(6) IR Panel	Receives signals from the remote control.
(7) Status LEDs	These toggle on and off when a picture is being transmitted; otherwise they stay lit when the device is on.
(8) VGA Connector	The display device cable is connected here.(15pin-female)
(9) Power Supply Input	9V power supply adapter is connected here.

Product Setup Instructions

Please make sure that the Digital Picture Frame Interface box packaging includes the following items: picture box unit, 9V power adapter, software installation CD, and RJ-45 Ethernet cable.

1. Place the picture box unit on a flat, level surface preferably near a wall outlet, VGA monitor and Ethernet router. Using the supplied Ethernet cable connect one end to a spare port on a DCHP compliant router, and connect the other end to the Ethernet port on the right side of the picture box unit.
2. Making sure the power is off on the VGA monitor, connect the analog cable securely to the VGA connector on the left side of the picture box unit.
3. Finally, connect the AC power adapter to the back of the picture box unit, and plug the adapter in the wall outlet.

The unit will automatically power up and wait for requests on the network and from the push buttons or remote control. The picture box unit is designed to receive automatic configuration via a DHCP server that is built in to most standard 100/10 Base-T routers.

The *infrared detector* will work with any standard Sony brand controller. There are four control buttons that work on the remote: channel up (PWR), channel down (FUNC), volume up (NEXT), and volume down (PREV). See the "Product Use Instructions" section for a functionality description.

The *software installation CD* contains a simple self-extracting compressed file that can be extracted to any directory on the user's hard drive. The user should copy all the desired picture files to this directory before executing the program. The program must have the domain name or IP address of the Rabbit server in order to connect to the rabbit. In order to connect to the Rabbit server, execute the PC client program with the IP address as the first argument and the port nuMBer (39) as the second argument. The Rabbit will now be able to transfer pictures from the PC to the display device.

How to use your DiPFI

Now that you have successfully set up your DiPFI the following guidelines should help you operate it with ease.

Displaying your first Picture :

Press the Next$_{(2)}$ button on top of the box to tell your DiPFI to send the first picture to the VGA device you connected it to. Within seconds the full picture should be displayed on the screen for your viewing pleasure. If this is not the case, try using the Power button to switch the device on and off. If this does not solve the problem, perform the Set-Up instructions again or try some more troubleshooting steps (Page 7).

Changing the Picture :

The Next$_{(2)}$ and Previous$_{(3)}$ buttons will cycle forwards and backwards through all the picture files in the DiPFI_PIC folder on your PC, displaying the full picture before recognizing the next command. Hitting Next or Previous while a picture is loading will not interrupt the current picture being sent to the display hence holding these buttons down will have the same effect as pressing them once.

Album Slideshow :

This display mode lets you view all the pictures in your DiPFI_PIC folder without having to hit next or previous buttons repeatedly which can be tedious. Pushing the Function$_{(4)}$ button once will continuously cycle through all pictures displaying them for a period of 5 minutes each. To stop on a certain picture push Previous and to exit this mode and return to the static display push Function again.

Remote Control:

With any SONY™ remote, you can control the DiPFI from up to 10m (33 feet) away within line of sight of the IR Panel. The functions work exactly as described above for the on-device push buttons. Programming may be required with any other remote protocol to implement the same functionality.

For Best Use:

- Use this product in a cool and dry area away from direct sunlight.
- Place product on a flat surface with the push buttons and connectors unobstructed; preferably a place where the Ethernet cable will not obstruct or impede movement.
- The IR panel on the front should be facing the area from which you would like to remotely control the display and should not be obstructed.
- Do not place objects on top of this device as inner components may be affected.
- Turn off the device when not in use.

Product Troubleshooting Instructions

Frequently Asked Questions:

1) The PC client software will not connect to the picture box unit. The client returns the error "Unable to connect to port 39 on xxx.xxx.xxx.xxx" –

The router must be configured to forward port 39 on the IP address assigned for the picture box to work over the Internet. Also make sure that the network cable is securely plugged into the picture box unit.

2) There is nothing displayed on the monitor –

The picture box unit requires that the PC client send it a picture before it can display anything. Make sure that the PC client successfully connects to the picture box unit.

3) The remote control does not work with the picture box unit –

Ensure that a Sony Remote Control is used, there are new batteries in the remote and there is nothing between the remote control transmitter and picture box IR Panel (9).

4) I have a problem not included in this F.A.Q. –

Feel free to call the Digital Picture Frame Interface technical support nuMBer at 1-800–GO-DiPFI (800-46-34734) from 9am – 7pm Monday through Saturday.

Any comments or suggestions are welcome and would be appreciated.

Contact Information:

Product Development Website: http://shay.ecn.purdue.edu/~477grp12/

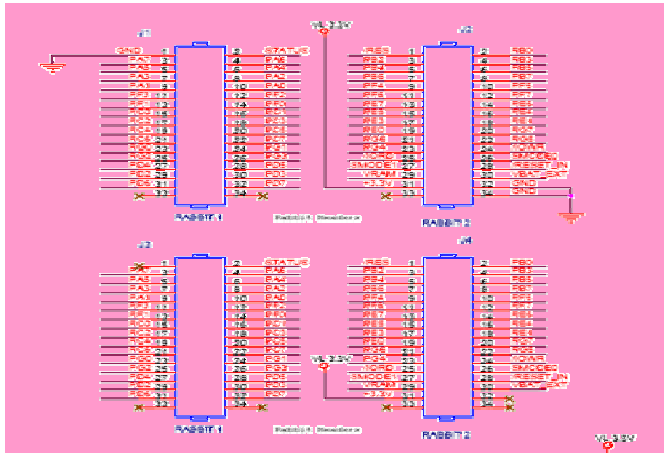
Email: 477grp12@ecn.purdue.edu

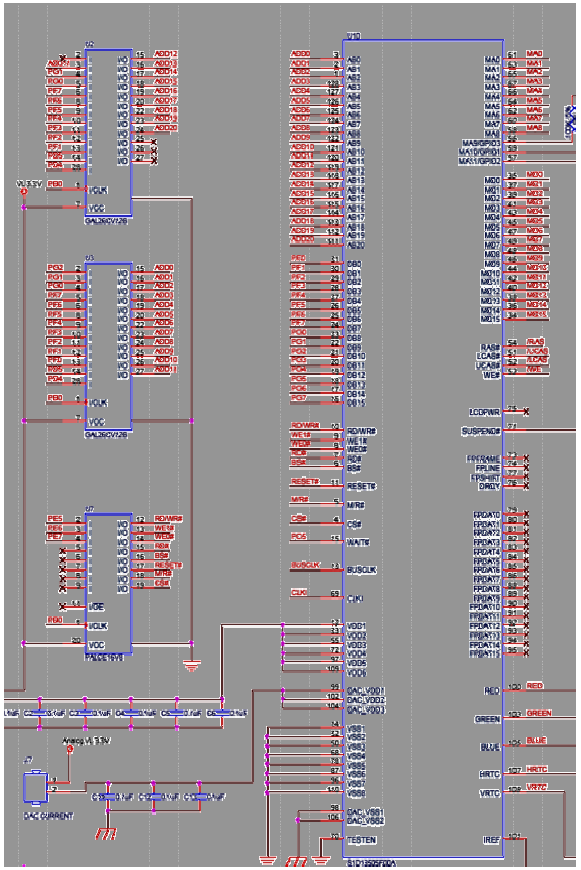
Appendix H: FMECA Worksheet

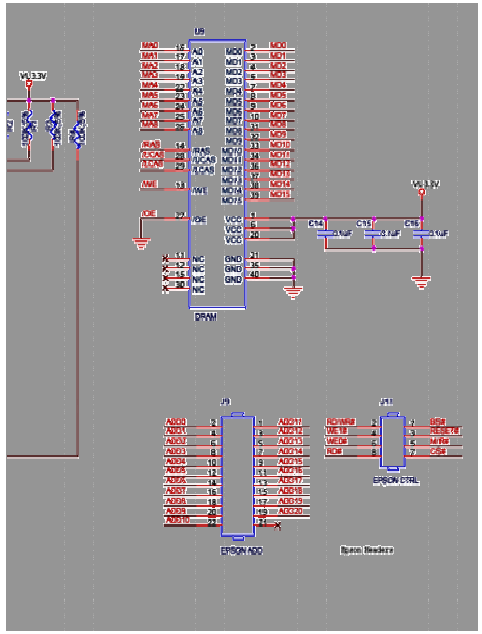
Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
Block A: Power Supply						
A1	Vcc = 0V	Failure of J6 or failure of U13, U14, U15 (Any component in Block A fails or an external short)	No power to U10, therefore no image displayed at all.	Observation	HIGH	
A2	Vcc > 5V	Failure of U13, U14, U15	Unpredictable effects	Observation	HIGH	
A3	Vcc out of tolerance	C7, C20, C8	High ripple or Operation at out of spec voltage; Unpredictable	Observation	HIGH	Monitor Wall Wart
Block B: User Interface						
B1	No response to pushbuttons	SW2, SW3, SW4, SW5	No new pictures obtained	Observation	LOW	Limits the users control method to IR remote
B2	IR fluctuating or corrupted	U16, U17	Random pictures displayed for undetermined periods of time	Observation	HIGH	

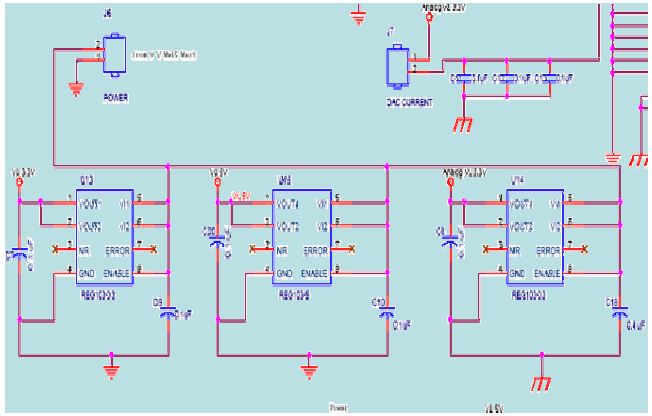
Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
Block B: User Interface						
B3	One of IR decoder outputs (PE0 - 4) stuck at 1.	U16,U17	Infinite cycle through images; Switching on and off continuously	Observation	HIGH	
Block C : Physical Connectors						
C1	All Outputs 0	U10, J8 R39,R40,R41, D5,D6,D7	No pixel data sent out on VGA connector	Observation	HIGH	Replace connector
C2	All Outputs 1	U10, J8 R39,R40,R41, D5,D6,D7	Wrong data on output connector; Unpredictable effects	Observation	HIGH	
C3	Random Outputs	U10, J8 R39,R40,R41, D5,D6,D7	Unpredictable effects	Observation	HIGH	
Block D: Graphics						
D1	Incorrect PLD outputs	U2, U3, U7	Pixel data sent to the wrong addresses, corrupted display	Observation	HIGH	
D2	PLDs Outputs all 0	U2, U3, U7, J5, U13	No addressing information available	Observation	HIGH	Bad Chip
D3	PLDs Outputs all 1	U2, U3, U7	Same Address is sent pixel data and is overwritten each time	Observation	LOW	

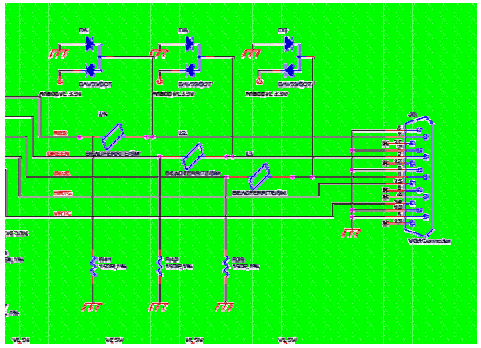
Failure No.	Failure Mode	Possible Causes	Failure Effects	Method of Detection	Criticality	Remarks
D4	EPSON all Outputs 0	U10, Q1, J20,U13	No meaningful data buffered in DRAM or sent to VGA output; Unpredictable	Observation	HIGH	User gets incorrect data from PC. Product is not useful. Easily detected.
D5	EPSON all Outputs 1	U10, Q1, J20,U13, C1 – C6, C11 – 13	No meaningful data buffered in DRAM or sent to VGA output; Unpredictable	Observation	HIGH	
D6	Clk always high (1)	U12	EPSON's communication and synchronization with rest of circuit is affected; can't interface easily	Observation	HIGH	
Block E : Microcontroller						
E1	All outputs 0	Failure in μC , U15, J6	NO functionality, interfacing, image transferring, No output at all	Observation	HIGH	Use reset controller
E2	All Outputs 1	Failure in μC	Unpredictable	Observation	HIGH	Reset μC
E3	Random outputs	S/W problem	Unpredictable effects	Observation	HIGH	Use headers to test S/W

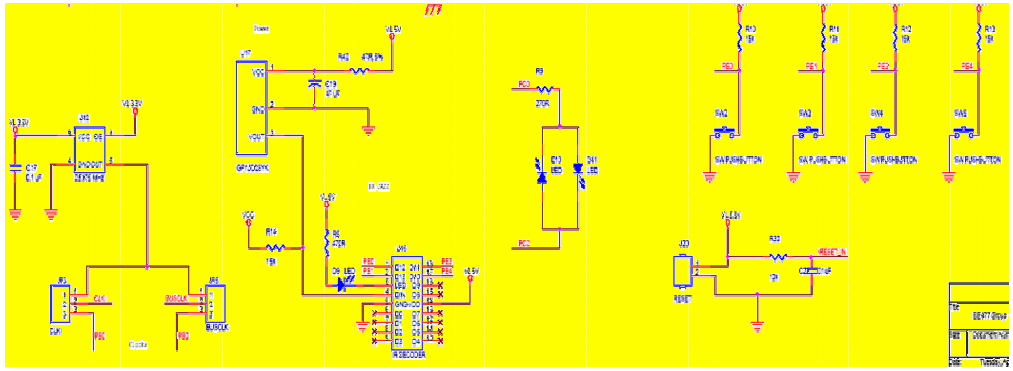












Power Supply	9V Wall Wart, Voltage Regulators
User Interface	Push buttons, IR Decoder and Receiver
Physical Connectors	VGA Connector, (RJ-45 on core module)
Graphics	EPSON, DRAM, PLDs, Crystal Oscillator
Microcontroller	Rabbit 3000 (Headers)