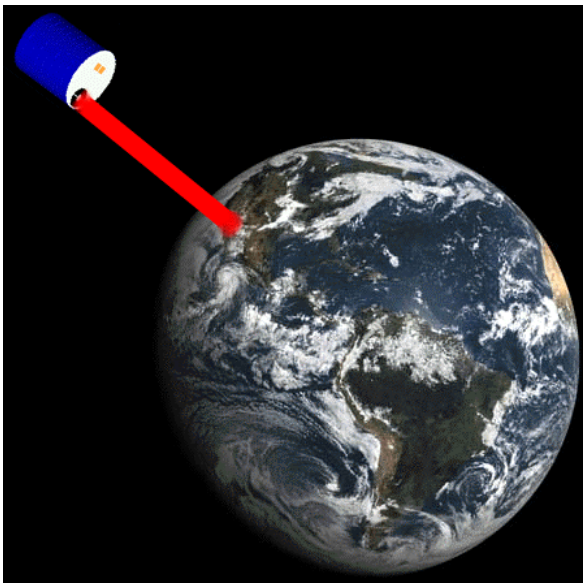
Manchester Encoding Test Link Interface for The Science and Technology Initiatives' Laser Communication Experiment for The Student Satellite Project's UASat



Prepared by: Christopher Gee
Matthew Gilbert

Submitted to: Glen Gerhard
Ed Lonsdale
Martha Ostheimer
Hal Tharp
William Wing

ECE 498
Department of Electrical and Computer Engineering
University of Arizona
Tucson, Arizona

Submitted April 27, 1998

Abstract

Satellite communication systems are typically based on microwave designs. These systems are usually power hungry, with slow transmission speeds. Conserving power on a satellite is an important goal of any communication system design. Although communications systems are necessary components, an efficient design reserves power for the scientific purposes of a satellite. This is especially true in the design of a laser based communication system for the University of Arizona's Student Satellite Project's UASat. Because the design of the entire laser communications system was too much to tackle in one semester, this project focused on the necessary interfacing for performing test links, which will help designers specify future system requirements. This project provides an ISA card / breadboard solution for getting data out of a computer, in the form of a Manchester encoded signal, and then decoding the data and getting it back into a computer. Flexibility and scalability of the test hardware were strongly emphasized, for the first iteration of the project. Future revisions of the hardware / software will combine everything onto a single ISA / PCI card solution.

Table of Contents

List of Figures

List of Tables

Introduction

One of the major obstacles faced by makers of satellites is the problem of power. A major use of power on a satellite is the communications system. Currently most satellite interaction is through microwave channels. These designs, however, use large amounts of energy. Current alternatives to power hungry microwave systems are radio wave links. These schemes, however, offer very limited bandwidth, which in many applications will not provide an acceptable solution. Satellite designs are wasting valuable space, time, and money on implementing these inefficient communication systems. A solution is to turn to lasers, which provide lower power consumption, higher bandwidth, and simpler electronics. The purpose of the Laser Communication Experiment for the Student Satellite Project's UASat, is to demonstrate the feasibility of laser communication link, on a small budget, short time frame, student driven level. A significant part of the design process is to use test links, that model the situations that may be faced in operation, and help refine design requirements. Also, a designer will want the ability to send a stream of data, have it go through a test laser link, and then come out on another computer.

Test links will provide a great deal of information for designers of the Laser Communication Experiment. Significant parts of the test link are the data processing and computer interfacing, which will allow a designer to easily experiment with different transmission mediums. There was no such hardware and software available, prior to this project, which provided designers with interfacing and data processing capabilities. In addition to not having the necessary hardware to produce Manchester encoded signals, the entire protocol layer was non-existent, and needed to be implemented in software. All this interfacing allows other system designers to connect amplifiers and drivers, which will modulate a laser and also amplify the output of a detector.

Since this was the first iteration of the project, there were many unknowns in the designed system. As a result, experimentation was done with two PC interfacing methods: serial and ISA bus cards. This led to crudely implemented hardware and software components. Although the simplest and fastest technique was utilized at every step, the best quality was still the end goal. For the hardware side, space and power consumption were not immediate concerns. On the software side, a rudimentary protocol was implemented. For the purposes of testing the basic functionality of the transmitting / receiving capabilities of the hardware, simple computer programs were written, to transmit, receive, and compare the data.

One difference that a reader might notice, between the Design Proposal for this project, and the actual technical report, is a shift in the purpose of the project. Initially, the goal was to design actual hardware and software that would be used for the real satellite. Instead, after going through the past semester, the overall objective was changed to providing the interfacing that would be necessary to conduct test links. Within this interfacing would be data processing and error correction / detection that would be similar to what would be used on the actual satellite. In other words, the test link interfaces developed in this project would provide a mirror of the functionality of the real system. Whether or not the designed hardware and software can be adapted for use on the real satellite, remains to be seen in future work.

Background on current test systems

Currently, most free space laser communication projects between an Earth based groundstation and an orbiting satellite rely on very simple protocols. Experiments such as the Ground / Orbiter Lasercomm Demonstration Experiment [1] used Manchester encoding [2], along with error detection routines. The goal of the JPL / Japanese team was to learn more about the atmospheres' affect on laser transmissions. Currently, other projects are planed / being planned (like the European SILEX and AstroTerra in the United States [3]). The problems with these systems are high cost and incredible complexity.

Many ground based experiments have been conducted and commercial networking solutions are available from companies such as AstroTerra [3], but cost and power requirements are limiting factors in applying existing technologies for the laser communication experiment for the UASat. Although there are many commercial solutions and experiments taking place, little applies directly to the needed test link hardware and software for the Laser Communication Experiment. As a result, a specialized set of test hardware and software have been designed, implemented, and tested, for future use as part of a Laser Communication Experiment Test link. Results from the test link can then be used in further design work for the actual hardware / software for the real satellite.

Design Criteria for Test Link Interfaces

In designing the test link hardware and software, several important criterions were kept in mind. The most basic criterion was designing a system that could be used with a standard PC (Intel x86 based). The second criteria was creating a system that used the Manchester bit signaling encoding technique. The third criteria was making the system as modular as possible, allowing users to connect as many unique drivers and amplifiers as possible. From these three basic criterion, several sub-design criteria / requirements were realized, and used as a basis for driving the overall design. The overall goal of this project was designing and building a set of hardware and software tools, that would allow any member of the STI Laser Communication Experiment to perform tests and evaluate the performance of a particular optical test link

Operating Platform and Operating System

The choice of operating platform is a decision of what type of hardware should this test link interface equipment work on: Sun Workstations, SGI, DEC Alpha stations, or run of the mill PC's. Since this is a "low" budget laser communication experiment, it was important to develop for an inexpensive and abundant platform, that almost everyone has access to, Intel x86 (and clone) architectures.

As a result, a decision was made in the design proposal stage to make whatever test hardware compatible with any type of PC that has ISA, PCI, and serial ports. Almost every PC and PC Clone available on the market has the industry standard ISA bus. A decision to develop for the ISA bus was also based on several technical sources on the ISA bus [4]. Choosing the Intel x86 architecture allowed for the ability to dramatically improve performance, and reduce development time by using faster hardware (CPU, memory, hard drives).

The choice of operating system was not as simple. Several alternatives were available: DOS, Windows95 (3.11 or NT), and Linux. Since software development would be taking place in C or C++, it would be natural to use Linux (since it comes with a number of free programming compilers). Also inherent in Linux was a robust set of networking tools, which would allow for remote operation of computers, simply by logging into a machine. Also, based on the availability of Real time Kernels for Linux, easier system management could be done, in a more robust environment than Windows or even DOS. Everything points toward more plusses with Linux (more powerful operating system for the x86 machines available, wealth of programming tools, system stability, and advantages of a UNIX environment), than the minuses associated with DOS or Windows 95/3.11/NT (difficult device driver creation, lack of "free" programming tools, and less capability for remote operation).

Provide Manchester encoding and decoding

Manchester encoding / decoding is a special way of encoding digital data by representing a '0' bit by a +voltage for the first half of the bit time, and –voltage for the second half of the bit time, '1' bit by a –voltage in the first half of the bit time and +voltage for the second half of the bit time. An extremely useful reference for Digital Encoding is available from Ronen Halevi and Udi Nir [5]. The implemented

design for the test interface links was created around a series of IC's, from a commercial manufacturer that automatically provided this functionality.

Associated with the specification of Manchester encoding / decoding was an implied data transmission speed of 10 Mbps. All design work has been done with this fact in mind, with the possibility present that even higher data transmission rates may be incorporated into future designs.

Modularity of the Hardware

In order to provide for the greatest level of functionality, it was important to use as many standard IC's and signal levels (TTL or RS-232) as possible. The input / output to the entire test link interface system was designed to be a single wire. Having a single input / output wire (instead of a parallel set) makes the interfacing job very simple. To get information from a computer through the test link interface, a step-by-step design process was used: starting with a simple RS-232 port and increasing the level of complexity to an ISA card). Important components that have to be part of the system design: a standard computer interface (ISA/PCI card or RS-232 compatible port), connection from the card to the data processing hardware, and a connection from the data processing hardware to the "real world" (or laser modulation / demodulation electronics). Designing and building the hardware for this project provides a jumping point for later project groups to improve on the hardware and eventually customize it for the satellite.

The hardware must also provide suitable error correction and detection, with as much forward error correction as possible. Implementing this design was not possible, but a rundown on what will be incorporated into the error detection and correction circuitry will be given.

Interface Requirements to "real world" electronics

Since this project is only part of the entire test link system, the output has been designed to comply with the TTL logic standard, which other designers will be using to design their amplification and modulation / demodulation hardware, which will interface to the laser. In addition to being TTL level signaling, the output stream, after Manchester encoding, was designed to utilize a single wire. The same is true for the input into the test interface: signals were assumed to be a TTL level, Manchester encoded signal on a single wire, which would be decoded, and a serial stream would be sent to the host computer, to be processed.

Modularity of the Software

The objective of the software design was to provide some sense of a protocol, for any users conducting test links. In the OSI 7 layer model of a network, the software that was designed and written, mimicked the behavior of the data link layer. As a result, all the software must provide are functions which allow programs at a higher level to send data through the test link interfaces. This data will then be sent out on whatever physical medium is below the protocol. The opposite process must be true also: the physical layer can either call a function to tell the protocol that data is present, or the protocol can ask the physical layer if there is any data present. The beauty of the layered model of the system is that tasks are separated and easier to implement in a modular fashion. The software developed in this project must also provide some type of buffer and flow control (it not implemented in hardware), and the capability of having a variable bit rate.

System performance measurement metric

A simple metric for measuring the performance of the test link interfaces was to count the numbers of correctly received data bytes and number of incorrect data bytes. The tests performed utilized two bytes of data and two bytes of preamble information (to help provide synchronization for the Manchester chips. The simple data that was sent was every number from 0 to 255, encoded as a hexadecimal number. In addition to going through the entire range of numbers, each number was repeatedly sent 10,000 times, to

provide a fairly large sample size. Results from these tests were tabulated in a simple text block, after each type of data was sent.

Space and power requirements for the test link interface prototypes

Since this was only a semester long project, space for mounting IC's and supplying power to them, were not immediate concerns. An attempt was made to layout IC's on the breadboards in a logical fashion, which allowed for easy interpretation of data flow. Developing on breadboards also allowed for multiple designs and easy corrections. For the final test link interfaces (beyond version 1), an ISA card can equipped with all the necessary lower power components on it, with no additional hardware (other than the interface cables to get signals into and out of the system). Power was not an issue, because a standard, dual lab bench power supply was used, that was capable of driving all the circuitry.

Design

Design Process

The design of the Test link Computer to laser Interface was broken into four steps. The first was to create a Manchester encoding receiving/transmit pair. The second step was to design and test a simple send and receive circuit using the serial ports on a computer. The third step was to design and test the circuitry used to interface with the ISA data bus on a computer. The fourth step was to design an interface between the Manchester circuitry and the ISA data bus.

Manchester encoding decoding

The first step in the design process was to create a Manchester encoded signal. It was determined that the easiest and most cost efficient method would be to obtain a commercial integrated circuit to accomplish this task. Although there are many products that perform this function, few met our requirements. Currently, most Manchester circuits provide additional functions beyond the encoding and decoding. For example, most performed the physical layer for Ethernet in addition to the encoding and decoding. For our application this was inappropriate because of the specialized functions we needed it to perform. We also required that the integrated circuit be able to perform at a minimum speed of 10 megabits per second (Mbps). The chip chosen was the National Semiconductor DP8391A Serial Network Interface.

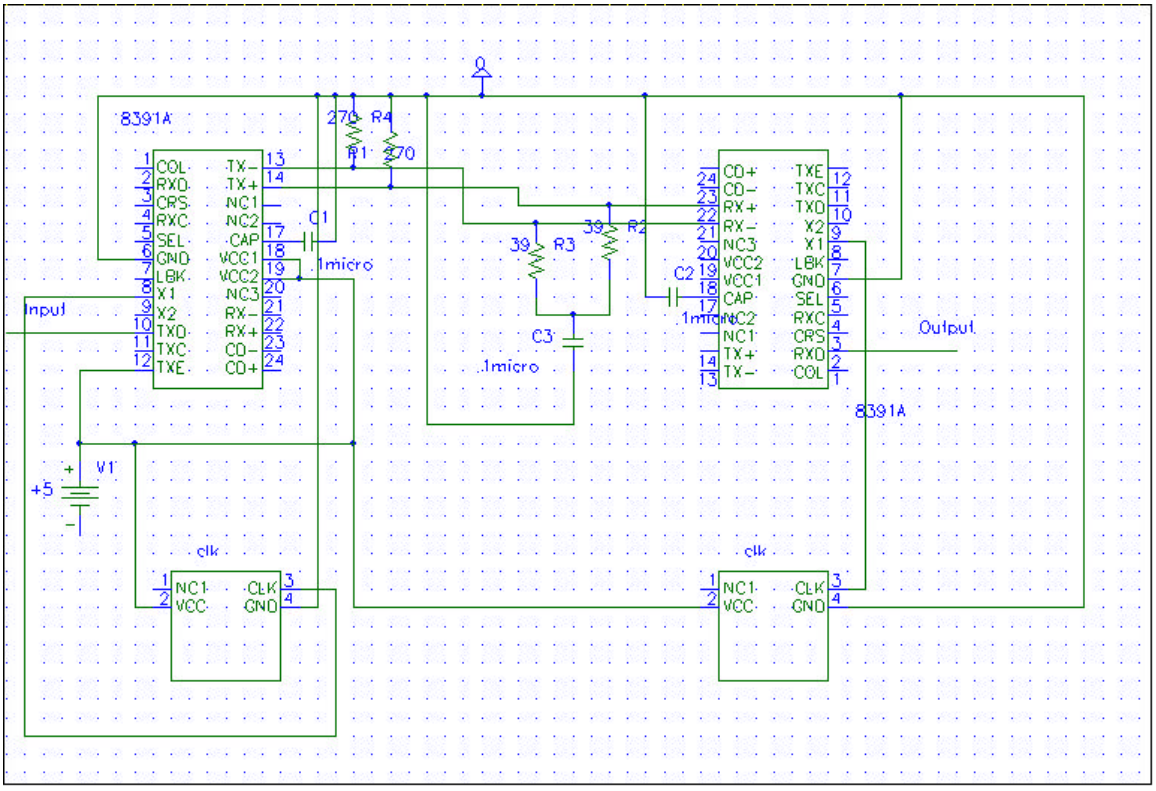
Our first design step was to set up a minimal circuit in order to test the functionality of the 8391. The expected functional operation of the 8391 was simply to be an interface between non return to zero (NRZ) data and Manchester encoded data. On the transmit side this was to accept NRZ input data and convert it to a Manchester encoded differential signal. On the receive side this chip was to accept differentially encoded Manchester data and convert it to NRZ data and clock pulses. This circuit can be seen in figure 1 along with it's pin-out in table 1. For this simple circuit only the necessary pins were used. This circuit behaved as expected most of the time. However, the circuit would experience random bit flipping errors where a one would be converted to a zero and vice versa.

Table 1. Pin-out of necessary Manchester pins

Pin	Name	Input/output	Description
2	RXD	Output	Output NRZ signal from the encoded data.
6	GND	N/A	Ground
8	X1	Input	20Mhz clock required by this chip.
10	TXD	Input	Input NRZ signal to be encoded by the 8391.
12	TXE	Input	Transmit enable. 8391 will only send data when this signal is

			high.
13/14	TX- TX+	Output	Differential line driver which sends the encoded data to the transceiver.
18/19	VCC	Input	Positive supply pins.
21/22	RX- RX+	Input	Differential receive input pair from the transceiver.

Fig. 1 Simple Manchester encoding/decoding circuit



Serial Port (Simple Send and Receive)

The purpose of the simple send and receive circuit, using the serial ports, was to create a very simple test link. The ultimate goal was seeing the 8391A's behavior in a real world situation. While the data rate being used was much lower than the target data rate, it was a good starting point from which future circuits were based. One of the major obstacles in getting this portion of the test link to work was converting the RS-232 level signals from the serial ports to the TTL levels required by the Manchester chips. Initial attempts to design these circuits are shown below. The circuit to convert an RS-232 level signal to TTL is shown in figure 2. The circuit to convert a TTL level signal to RS-232 levels is shown in figure 3. However, major problems were faced when trying to integrate these circuits with the rest of the Satellite due to the required negative voltages. Therefor, an integrated solution was found from Analog Devices. The ADM223 is an RS-232 driver/receiver. This circuit needed only one 5 Volt input and therefor met the needs of our project much better than the previous solutions.

Fig. 2 TTL to RS-232 circuit

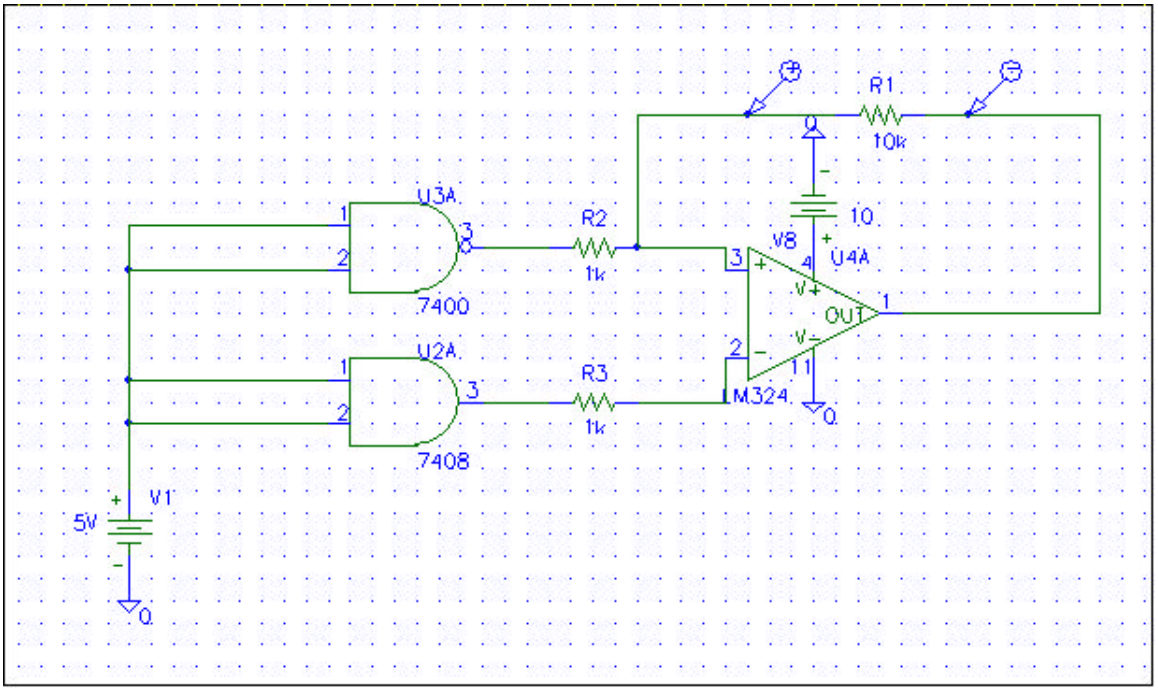
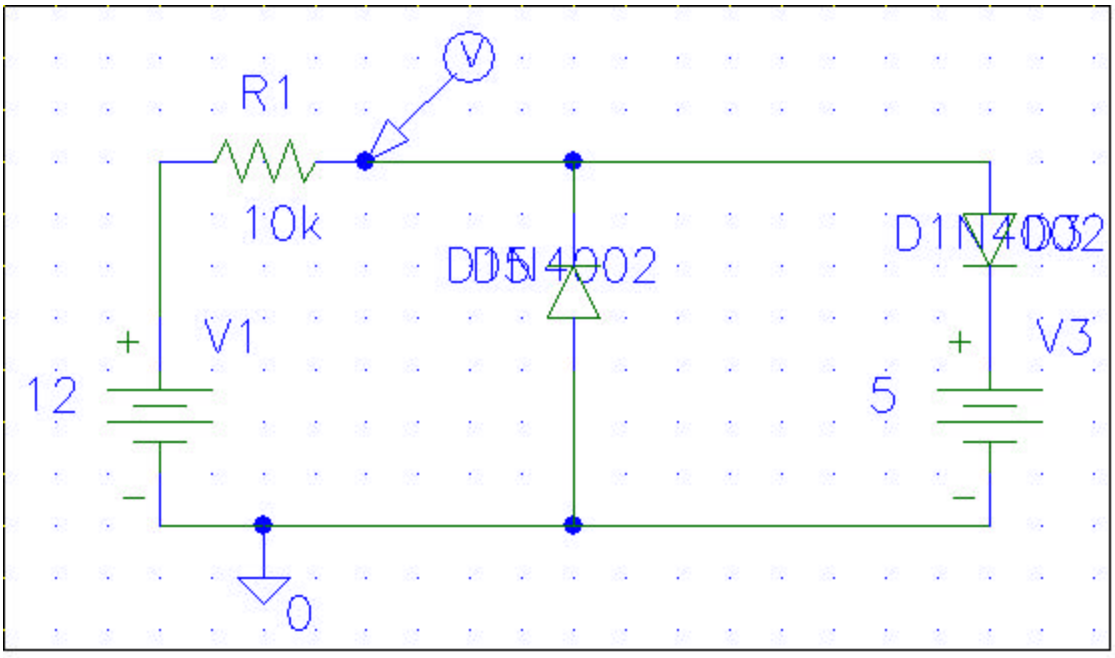


Fig. 3 RS-232 to TTL level circuit



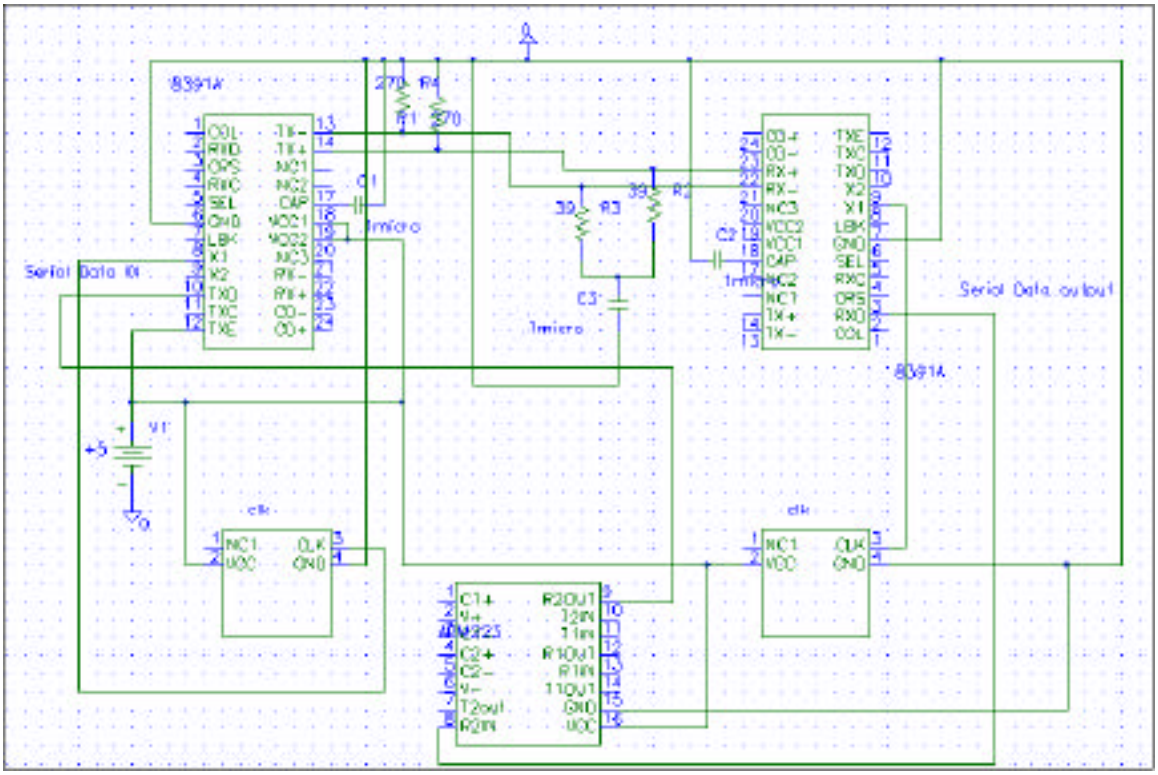
In order to use the serial ports on the computer we needed to construct a null modem cable. The pin-out for the null modem cable is shown in table 2.

Table 2. Pin-out for null modem cable

Pin	Name	Input/output	Description
1	PG	N/A	GND
2	TD	Input	Serial Transmit data. This input was put into pin 10 of the 8391.
3	RD	Output	This came from pin 2 of the 8391.
4	RTS	Input/output	Was put into pin 5 of the other computer.
5	CTS	Input/output	Was put into pin 4 of the other computer.
6/8	DSR DCD	Input/output	Was put into pin 20 of other computer
7	SG	N/A	GND
20	DTR	Input/output	Was put into pin 6/8 of the other computer.

Using the ADM223 the following circuit was constructed and tested (figure 4).

Fig. 4 Serial data connection circuit



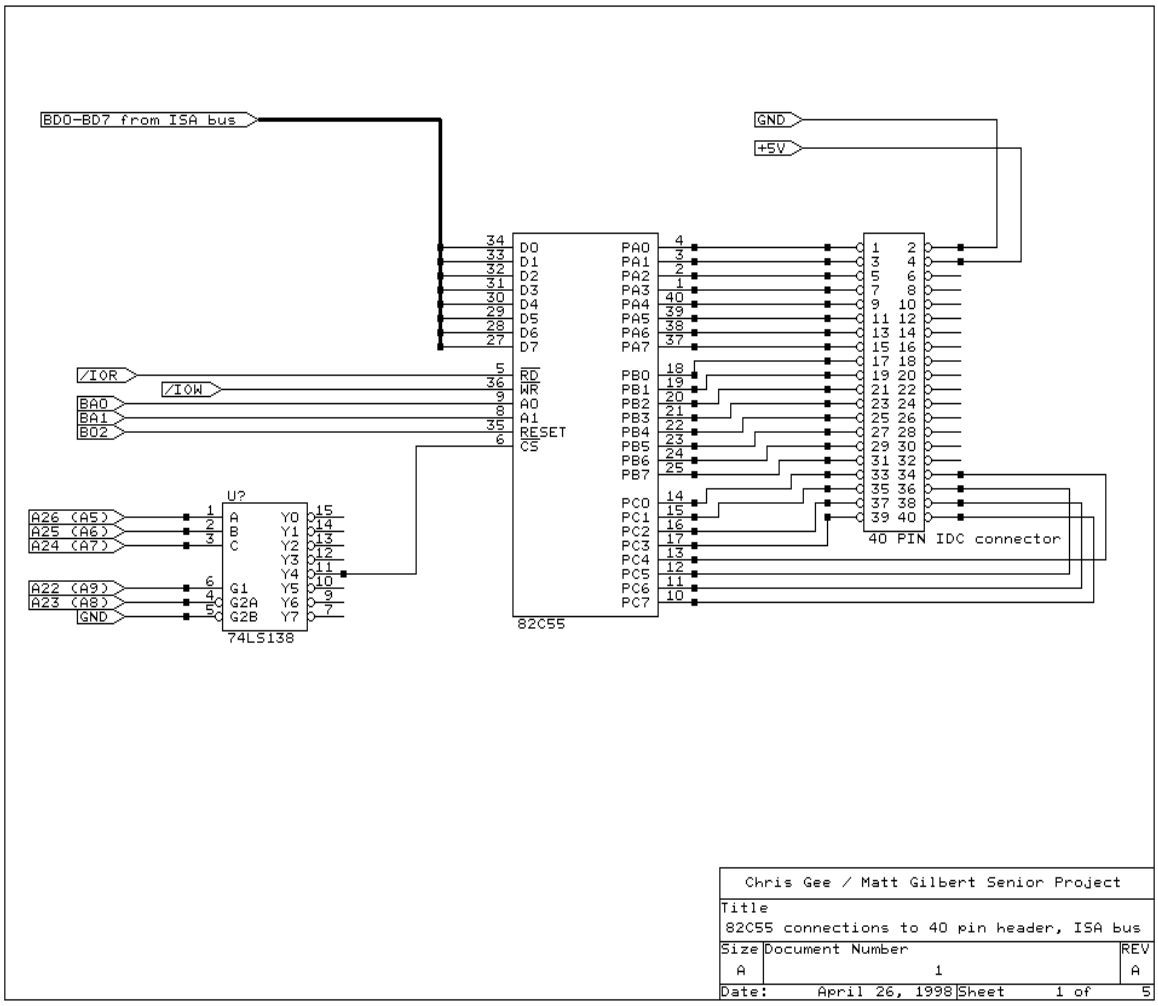
All the capacitor connections shown on the ADM223 in figure 4 were implemented internally by the chip which was a later revision of the circuit. Using the circuit constructed above, a reliable communications link was created through the 8391A's. The bit flipping errors encountered in the previous circuit (figure 1) were not seen. It was from this basic idea that future circuits were created.

Interface to the ISA bus

The next step in the design process was the construction of an interface to the ISA bus. The ISA bus was the connection between our test link circuitry and the computers' microprocessor. The decision to use the

ISA bus was based on its relative ease of use and its speed was sufficient for our application. The interface was chosen to be the Intel 82C55A. This chip has 24 programmable I/O pins and is compatible with all Intel microprocessors. It was chosen because of its ease of use and its functionality. In order to interface this chip with the microprocessor it was necessary to mount it on an ISA card. This was constructed and tested by doing simple reads and writes to the registers in the 8255. The connections between the ISA bus and the 8255 are shown below in figure 5.

Fig. 5 ISA bus interface



As seen in the circuit above, minimal supplementary circuitry was needed to interface to the ISA bus. The only addition was an address decoder. This allowed our circuit to operate along with all other peripheral devices in the computer. Address pins 8 and 9 were connected to “chip select” on the 74LS138. Address pins 5, 6, and 7 were connected to the inputs of the 3 to 8 converter to allow us to select the peripheral address of our choice. Using this addressing technique, valid I/O addresses ranged from 200h to 2E0h. The I/O pins of the 8255 were then connected to a 40 pin connector which would be used to attach to the external circuitry. The decision to place all other circuitry off of the ISA card was made because of the difficulty in changing pin connections when wiring wrapping. Wiring wrapping is necessary to connect any integrated circuits to the ISA card and therefor to the ISA bus.

The important results from this circuit were a working interface to the microprocessor. We were able to read and write data bytes to and from the 8255’s control and data registers.

Interfacing the Manchester chips to the ISA bus

The final step in the design process was to construct an interface between the ISA bus and the Manchester circuitry. This was by far the most complicated design process yet performed. The biggest difficulty was converting the parallel data from the ISA bus to a serial data stream to be used by the 8391A. The difficulty of this task was due to the precise timing requirements needed to perform these operations. Another problem was that the 8391A's produced a differential output. This was not suitable for our situation since the Manchester encoded signal needed to be sent through a laser. The first step in solving this interfacing problem was to convert the differential output into a single line.

Before anything else could be done, the differential outputs of the 8391A had to be examined. The results were unexpected. Small differences were noticed on the two differential output lines between inputting 0's and inputting 1's. These differences were at most .6V. On a 5 volt signal this did not appear to be substantial. However, a differential line driver was purchased and was able to distinguish between the small differences and produced the correct output. The chip used to accomplish this was the National Semiconductor DS26LS32C quad differential line driver. On the receiving end the single signal needed to be converted back to a differential signal for the 8391A. Initially a 7404 inverter was used. The 7404 produced a voltage level too high from the signal being transmitted and therefore did not work. When the 8391A subtracted the two differential signals, there was too much of a DC offset for it to tell the difference between a 0 and a 1. To fix this problem a differential line receiver was purchased. The chip chosen was the counterpart of the differential line driver, an SN54265 quad differential line receiver. This chip produced two differential signals using the same voltage levels for each, unlike the inverter, which fixed the problem.

Serial and parallel data transformations

The next step in designing the interface was to create the circuitry to transform a byte of data into a serial stream to be sent through the Manchester chip. In order to decide how to transform this information a decision was made on how to construct a hardware data packet. It was decided that a 16 bit preamble and a 16 bit data word would be used. This resulted in the packet shown in figure 6.

Fig. 6 Physical medium packet

8 bits of preamble	8 bits of preamble	8 bits of data	8 bits of data

The next step was to determine all of the control signals that would be needed between the Manchester circuitry and the 8255. Port B was chosen to perform the control signals. Port B on the 8255 is BASEIOPORT plus one where the BASEIOPORT is the I/O port location defined above by the addressing logic. These control signals are summarized in table 3.

Table 3. Control signals

Control signal	Pins of port B	Description
Latch Select	0-2	Used to select memory latches.
Counter reset	3	Used to reset the receive data counters.
PL	4	Used to initiate the send process.

To perform data operations, port A of the 8255 was chosen to be a bi-directional bus. Port A on the 8255 is BASEIOPORT. In order to use the 8255 in this manor, it had to be put into operational mode 2. This

was performed by setting the first two pins of the control register on the 8255. The control register is located at BASEPORT plus three.

In order to transform the parallel data from the ISA bus to a serial signal, shifters were used. Four eight bit parallel to serial converters, or shifters, were used to send all 32 bits of the packet in figure 6. In order to only send 32 bits of data logic was constructed to control the transmit enable pin on the 8391A. In order to send the correct data, transmit enable went high under the following conditions: when PL was set and when the counters were less than 32. In order to count to 32 two four bit counters were used. The transmit enable logic is shown in figure 6.

To provide the correct data to the parallel to serial converters, only two eight bit latches were used. To reduce the costs of the test link, it was decided that the preamble would be hardwired to the shifters. In order to send a constant stream of data the shifted output of the parallel to serial converters was fed into the serial input of the next parallel to serial converter. The same transmit enable logic to control the 8391A was also used to control the parallel to serial converters. The parallel to serial converters would shift out the information from their parallel inputs followed by the information read in on the serial input. It was through this function that a constant 32 bit serial stream was created. The transmit process is shown below in table 4. The circuitry used to perform this process is shown in figure 6.

Fig. 6 Circuitry for transmit process

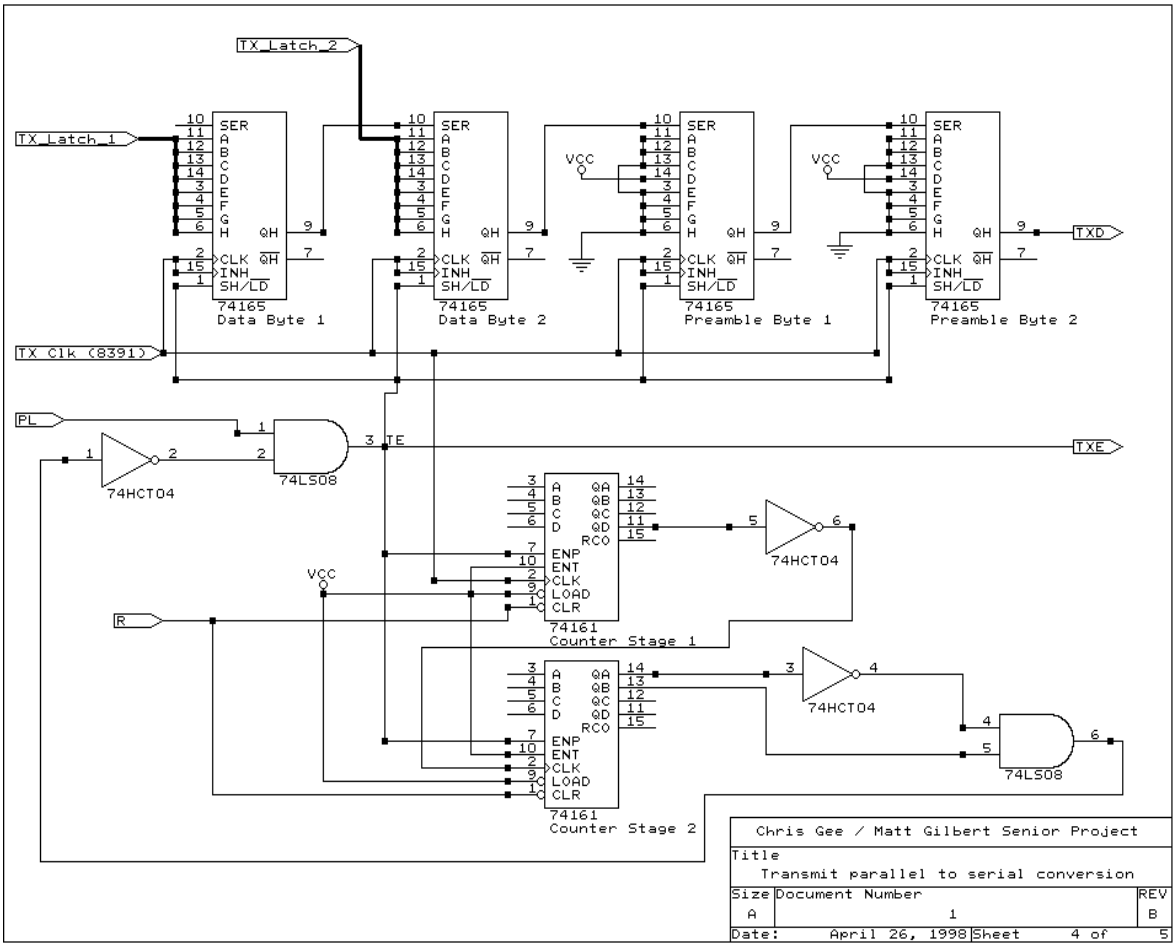


Table 4. Transmit process

Step	Description
1	Enable the first transmit latch by writing to port B.
2	Write data to the latch through port A.
3	Enable the second transmit latch by writing to port B.
4	Write data to the lath through port A.
5	Reset the counters by setting bit 3 on port B low then high.
6	Enable the PL bit to send transmit enable high

The receive process was more difficult than the transmit process due to the timing issues. When the 8391A detects a valid incoming signal it starts the receive clock, which is a 10Mhz signal corresponding to the data rate. It was through this clock that serial information was shifted into the serial to parallel converters. Three 8 bit serial to parallel converters were needed even though only 16 bits of data were expected. This was due to a specification of the 8391A which kept the receive clock running for five clock cycles after valid data had stopped. Therefore, the first input latch was connected to the last three bits of the first shifter and the first five bits of the second shifter. Likewise, the second receive latch was connected to the last three bits of the second shifter and the first five bits of the third shifter. By placing the data at the end of the packet it was irrelevant how long the preamble was. This decision was made in order to allow variations in the amount of time it took the 8391A to lock to an incoming signal. Because there was no buffering for this circuit, a simple polling I/O program was insufficient. Therefore logic was created to produce an interrupt to the microprocessor. The receive process is shown in table 5 and the circuitry for the receive process is shown in figure 7.

Fig. 7 Circuitry for receive process

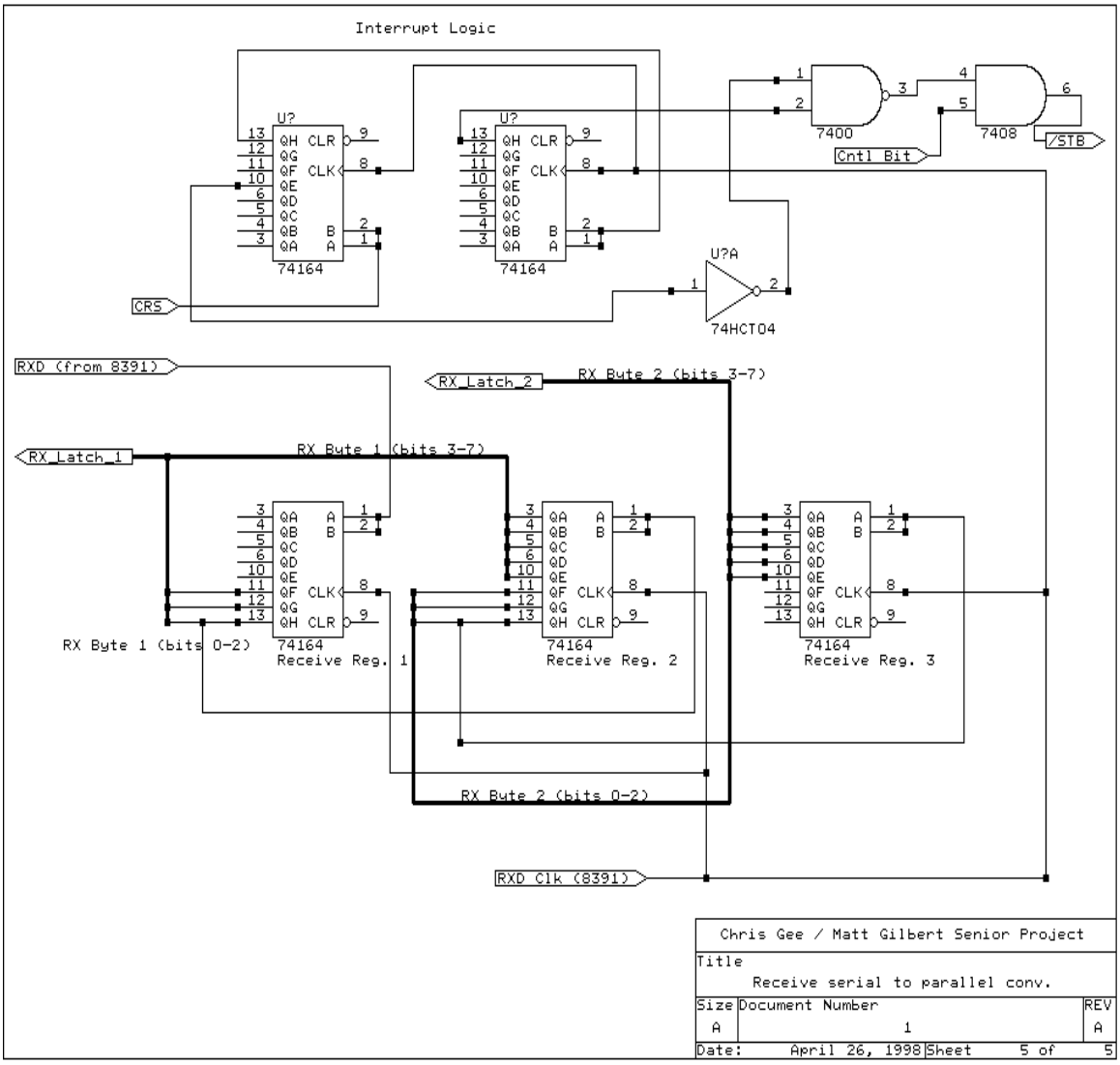


Table 5. Receive Process

Step	Description
1	Data appears on 8391A
2	8391A generates the receive clock which begins shifting data into the serial to parallel converters
3	Carrier sense goes high producing an interrupt
5	Interrupt routine chip selects the first receive latch.
6	Reads the data from the first receive latch.
7	Interrupt routine chip selects the second receive latch.
8	Reads the data from the second receive latch.

Due to the complexity of wiring up all of the chips needed for this process, multiple wiring mistakes were made. Design errors were also made when creating the counting mechanism for the receive cycle. The initial wiring for the incorrect counters can be found in the appendix A.

Design Testing

Simple Manchester encoding/decoding testing

To test the simple encoding/decoding circuit, seen in figure 1, manual transitions between high and low were performed. This was done by physically moving a wire connected to TXD from high to low. This was suitable at this stage because of the relatively minimal goals. Bit flipping errors would occur when the input was connected to ground and a high was received or vice versa. Currently there is no explanation for these random occurrences.

Serial input/output testing

The testing of the circuit seen in figure 4 was done using a program called miniterm. Miniterm is a program which is run on the Linux operating system. It was written by Sven Goldt and is freely distributed with the Linux Programmers Guide. The code for this program can be found in appendix A. The circuit was tested using this program by sending characters across the serial ports. Validation of a reliable signal was seen if the sent character was also received. During initial testing of this circuit, this was not always the case. However, after incorrect wirings were corrected, the 8391A's provided a 100% reliable communication stream.

Interface to the ISA bus testing

The testing of the 8255 was extremely simple. It consists of the short code sample found in the Appendix A and the correct wiring to the ISA bus. Luckily, there were no incorrect wirings on this portion of the test link and testing was extremely easy. To validate the correct performance of the 8255, bytes of data were written to the registers in the circuit and then were read back out. If the values matched the circuit was working.

Serial and parallel data transformation testing

To test this final integration of previous circuits, everything was wired to the same computer. In this manner it would be receiving the information it sent. Currently, none of the interrupt routines are wired or have code written for. Therefore, to test this circuit a simple polled I/O method was used. Although this would not work in a real situation, for testing purposes it was fine. The code for this portion of the test can be found in Appendix A. The test was performed by going through the transmit process followed immediately by the receive process and comparing the transmitted and received values. Unlike previous tests, a 100% reliable data stream was never achieved. After days of debugging the circuit it appears that noise was brought into the circuit, possibly by using breadboards, disrupted some of the timing routines. However, through extensive testing it appears that this circuit works 83% of the time. This result was obtained by sending values ranging from 00h to ffh with each value being sent 10,000 times. Remarkably, the random timing errors described above do not appear as random, which is illustrated in the graph below. Future experimentation with this circuit will be needed to fix the timing problems to create a 100% reliable transmission stream.

Experimental Results

As mentioned in the design portion of the report, testing was performed by simply sending and receiving data through the single prototype card that was constructed. A count was made of the number of correctly received data bytes and incorrectly received data bytes in 10,000 attempts. The data byte being referred to consists of two bytes, of the same number, that were sent together, one after the other. Please refer to the test program in Appendix A. Graph 1 shows the percentage of correctly received data bytes, per data byte that was transmitted. Graph 2 shows the percentage of incorrectly received data bytes, per data byte that was transmitted

Fig. 8 Plot of number of correctly received data bytes

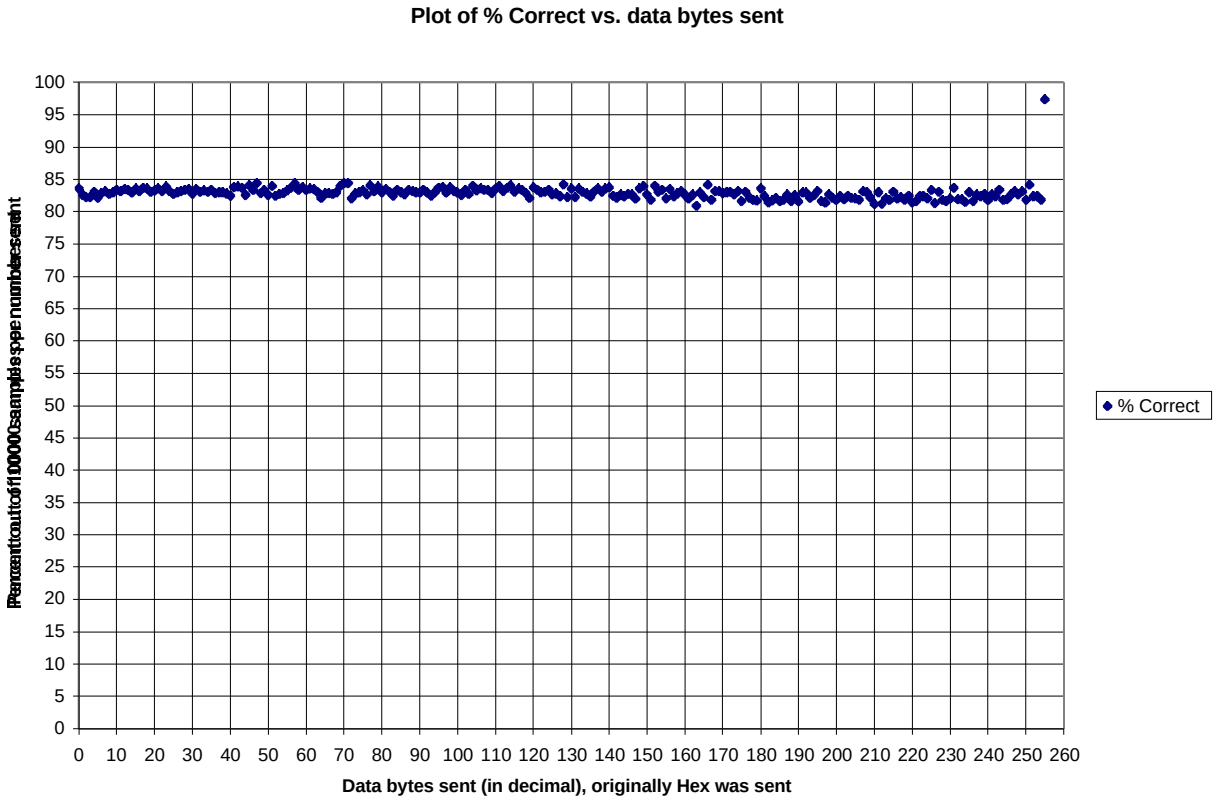


Fig. 9 Plot of number of incorrectly received data bytes

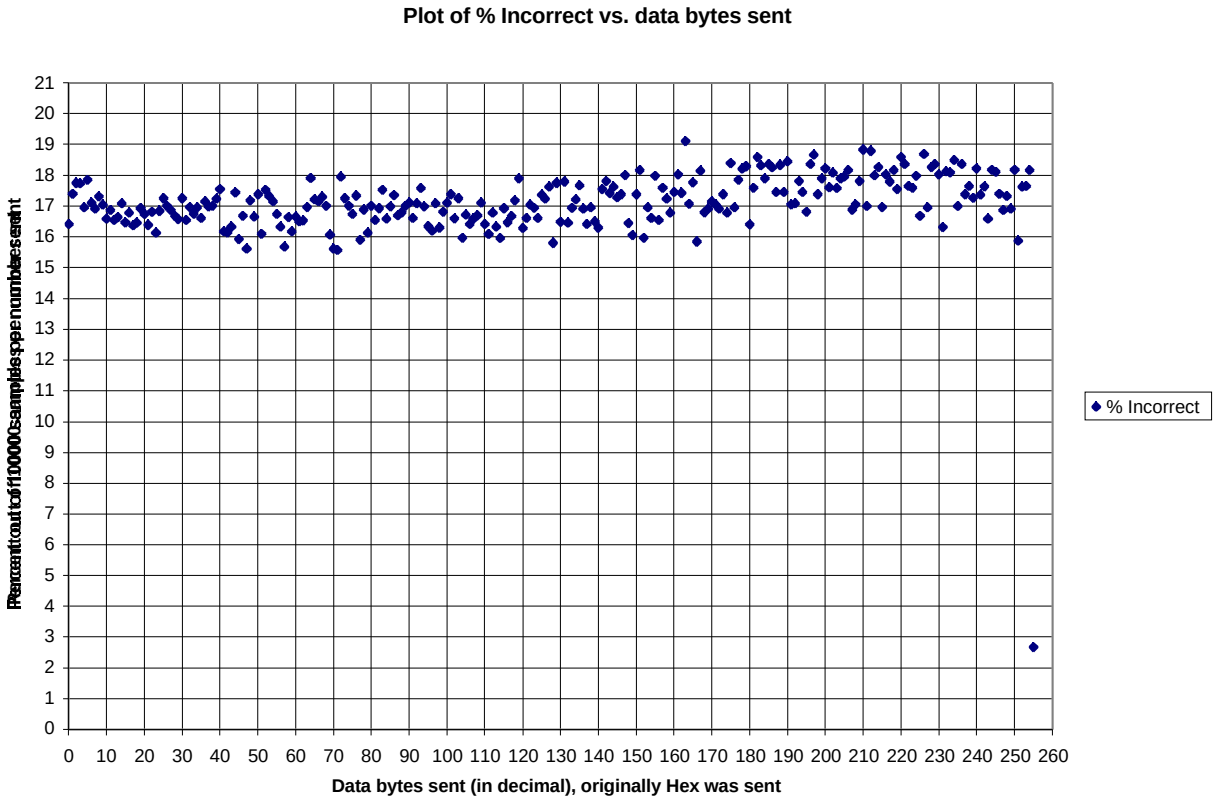


Table 6. Average, Median, Mode, Min and Max of the test data

	Average	Median	Mode	Min	Max
Correctly received data	82.88629	82.925	83.05	80.89	97.33
Incorrectly received data	17.11367	17.075	16.95	2.67	19.11

Actual data is available in Appendix B.

Conclusions and Recommendations

Conclusions on Design and Implementation

This project started off with the problem of designing a system of hardware and software to provide a "service" to the test link designers: an interface to allow them to perform tests by sending and receiving data. The system that has been designed and implemented has several attractive features. By using Linux and networking the two x86 based computers (that form the ends of the link) with standard Ethernet, users have the ability to do remote management of one or both of the machines. Another advantage of the implemented hardware and software is that this is the first step in starting up test links for the Laser Communication Experiment for the Science and Technology Initiatives Team. Commercial solutions are

too costly to utilize, require too much hands-on, and the time frame for many of the Commercial research projects is on the scale of many years which is much too long for a student project which plans to take a few years per satellite.

Initially, the objective was to design the actual hardware and software, that would be used for data processing, error detection and control, buffer management, and interfacing to other satellite subsystems. Actual hardware being the Manchester encoding IC's, error correction / detection algorithms in silicon, and buffering provided by a microcontroller and memory. And the actual software that would have been developed would be a protocol stack, connecting the processes running on the satellite and the hardware that would be responsible for actually transmitting, receiving, and decoding the laser signal. Instead, the objective that was achieved was developing a computer interface to a test link, which used similar types of bit encoding, with the possibility of adding protocols on top of the existing software drivers.

Even though the objective for this project was met, there were a number of problems that arose. The working system had a fair amount of error in transmissions, as shown in the results section above. Assumptions were made that the breadboard environment and power supplies did not contribute a significant amount of analog noise. Even though most of the work was done in the digital domain, there were instances when analog problems arose, which were not accounted for. Capacitive coupling between wires was eliminate as much as possible by using twisted pair wires between breadboards. One breadboard was responsible for representing the Manchester encoding / decoding, while the other board was responsible for interfacing the ISA card to the Manchester chips, with a series of latches, parallel to serial and serial to parallel converters. Throughout the entire implementation process, a piece by piece construction process was used: starting with the RS-232 port interfacing and eventually working up to an ISA card and breadboard setup.

Comments on Results

A very simple metric for measuring the performance of the implemented system was developed. A simple count of the number of correctly and incorrectly received data packets, out of a total of 10000 tries was used. After examining the data, it seemed very strange that, in general, the percentage of correctly transmitted numbers hovered around 83%, with the corresponding error percentage of about 17%. Based on the design of the system, the expectations were for an extremely accurate system, with a very low error rate. This in fact happened for the case of sending a packet consisting of $FFFF_{hex}$, which resulted in an accuracy of about 98%. The preamble being used for all the testing was hardwired to be a set of three 1's (+5V), a 0 (0V), and 4 more 1's (+5V). It would make sense, for the case of sending $FFFF_{hex}$, that there would be few errors, because an almost continuous stream of data would be sent. There might be a strong dependence on the type of data that was sent, and the percentage of destroyed and incorrect packets.

This simple test and the results from them, lead to the conclusion that there must be some constant source of error in the design, which causes random bit flips or other synchronization problems. The tests were run in an automated fashion, and took a considerable amount of time to run on an 80386 machine. A simple "loopback" setup was used, where a signal would be fed out of the ISA interface card, routed through the Manchester board, and then brought back into the ISA card. The following day, after the tests were completed, an examination of the actual stream of data being shifted out was performed. When an error did occur in the packet of data, it was an extremely large error (either all the bits went high or low), completely destroying all data. This lead to the investigation of the entire system, with no immediate answer to the source of the errors.

Nonetheless, there are multiple uses for the equipment that has been developed. It may become part of the test link design or be adapted to become part of the actual satellite. The generic nature of the input into

the test link interface, from the standpoint of the computer, allows it to produce output which can be transmitted across any medium. Developing the system to rely on standard, industry accepted interfaces, like ISA and RS-232, make all the equipment portable. Development on Linux has also made the software code very portable.

Recommendations for future versions

Even though version 1 of the Manchester Encoding based, test link interface worked better than expected, there are still several areas which can be improved upon. One of the first is making sure everything fits onto a single ISA card, to eliminate potential sources of problems like long wires, interference and signal loss. In addition to putting everything onto a prototyping card, the possibility exists, after a more robust and finalized design are created, to etch an ISA circuit board and mount many of the components directly onto the card, avoiding the hassles with wire wrapping.

Although the 82C55A peripheral interface adapter was an adequate solution, there are certainly much better solutions available. For instance, a better choice in controlling the interface between the ISA bus and the Manchester encoding / decoding would be a specialized communications microcontroller, that has built in serial in / serial out capabilities. A large part of the design time with version 1 was spent, converting the parallel data coming from the ISA bus and 82C55A into a serial stream that the Manchester chips liked, and turning the serial data stream into an acceptable parallel data stream. A microcontroller would also contain some EPROM storage and memory which would allow for a significant part of the lower level code to reside on the card, and not burden the main processor.

To further help with the software interfacing, further work has to be done with writing a loadable device driver, that is compatible with the Linux 2.x.x kernels. At the time being, the multi-tasking scheduling system in Linux may push off the communication tasks with the ISA card to a pretty low priority level, since they are being operated as a user process. If this was instead moved into a part of the Linux kernel, more “attention” can be paid to the ISA card, improving the performance, and maybe reducing the number of synchronization errors, that could be caused by the processor ignoring the ISA card. All of these changes can help enhance version 1 of the Manchester encoded test link interface to make it a more powerful and useful piece of hardware for the Laser Communication Experiment design team and for the Science and Technology Initiatives Team.

References

[1] K. E. Wilson, “An Overview of the GOLD Experiment Between the ETS-VI Satellite and the Table Mountain Facility,” The *Telecommunications and Data Acquisition Progress Report 42-124, February 15, 1996, Jet Propulsion Laboratory, Pasadena, California.*

[2] K.E. Wilson, J. R. Lesh, K. Aroki, Y. Arimoto, "Preliminary Results of the Ground / Orbiter Lasercomm Demonstration Experiment between Table Mountain and the ETS-VI Satellite," *SPIE Proceedings*, vol. 2699, pp. 121-132, Jan. 1996.

[3] AstroTerra Corporation Information. Copyright 1996 by AstroTerra Corporation. Internet, http://www.astroterra.com.

[4] Shanley, T. and D. Anderson, *ISA System Architecture*. Reading, Massachusetts: MindShare, Inc., 1995.

[5] UNI. Copyright 1994 by UNI, Inc.. Internet, http://www-dos.uniinc.msk.ru/tech1/1994/digenc/main.htm.

Technical Data for components was gathered from the following specification sheets:

- National DP8391
- Intel 85C55A
- 74LS373
- 74164
- 74165
- 74LS138
- 74HCTLS04, 74LS08, 74LS00
- 74LS265
- 74161
- DS26LS32C
- SN54265 quad differential line receiver
- Analog Devices ADM223 RS-232 / TTL converter

Appendix

Appendix A: Code listings

Minterm.c code

```
/*
 *  AUTHOR: Sven Goldt (goldt@math.tu-berlin.de)
 *
 *  This program is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU General Public License
 *  as published by the Free Software Foundation; either version 2
 *  of the License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
*/
/*
 This is like all programs in the Linux Programmer's Guide meant
 as a simple practical demonstration.
 It can be used as a base for a real terminal program.
*/

#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>

#define BAUDRATE B1200
/* #define MODEMDEVICE "/dev/modem"  */
#define MODEMDEVICE "/dev/cua0"
#define ENDMINITERM 2 /* ctrl-b to quit miniterm */

#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

void child_handler(int s)
{
   STOP=TRUE;
}
```

```
main()
{
int fd,c;
struct termios oldtio,newtio,oldstdtio,newstdtio;
struct sigaction sa;

/*
  Open modem device for reading and writing and not as controlling tty
  because we don't want to get killed if linenoise sends CTRL-C.
*/
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* save current modem settings */

/*
  Set bps rate and hardware flow control and 8n1 (8bit,no parity,1 stopbit).
  Also don't hangup automatically and ignore modem status.
  Finally enable receiving characters.
*/
 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

/*
 Ignore bytes with parity errors and make terminal raw and dumb.
*/
 newtio.c_iflag = IGNPAR;

/*
 Raw output.
*/
 newtio.c_oflag = 0;

/*
 Don't echo characters because if you connect to a host it or your
 modem will echo characters for you. Don't generate signals.
*/
 newtio.c_lflag = 0;

/* blocking read until 1 char arrives */
 newtio.c_cc[VMIN]=1;
 newtio.c_cc[VTIME]=0;

/* now clean the modem line and activate the settings for modem */
 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);

/*
  Strange, but if you uncomment this command miniterm will not work
  even if you stop canonical mode for stdout. This is a linux bug.
*/
 tcsetattr(1,TCSANOW,&newtio); /* stdout settings like modem settings */
```

```
/* next stop echo and buffering for stdin */
 tcgetattr(0,&oldstdtio);
 tcgetattr(0,&newstdtio); /* get working stdtio */
 newstdtio.c_lflag &= ~(ICANON | ECHO);
 tcsetattr(0,TCSANOW,&newstdtio);

/* terminal settings done, now handle in/ouput */
 switch (fork())
 {
  case 0: /* child */
   /* user input */
   close(1); /* stdout not needed */
   for (c=getchar(); c!= ENDMINITERM ; c=getchar()) write(fd,&c,1);
   tcsetattr(fd,TCSANOW,&oldtio); /* restore old modem setings */
   tcsetattr(0,TCSANOW,&oldstdtio); /* restore old tty setings */
   close(fd);
   exit(0); /* will send a SIGCHLD to the parent */
   break;
  case -1:
   perror("fork");
   tcsetattr(fd,TCSANOW,&oldtio);
   close(fd);
   exit(-1);
  default: /* parent */
   close(0); /* stdin not needed */
   sa.sa_handler = child_handler;
   sa.sa_flags = 0;
   sigaction(SIGCHLD,&sa,NULL); /* handle dying child */
   while (STOP==FALSE) /* modem input handler */
   {
    read(fd,&c,1); /* modem */
    write(1,&c,1); /* stdout */
   }
   wait(NULL); /* wait for child to die or it will become a zombie */
   break;
 }
}
```

io.c code

```
       /*
        * example.c: very simple example of port I/O
        *
        * This code does nothing useful, just a port write, a pause,
        * and a port read. Compile with `gcc -O2 -o example example.c',
        * and run as root with `./example'.
        */

       #include <stdio.h>
       #include <unistd.h>
       #include <asm/io.h>
```

```
#define BASEPORT 0x280 /* lp1 */

int main()
{
  int i;
   /* Get access to the ports */
  if (ioperm(BASEPORT, 4, 1)) {perror("ioperm"); exit(1);}

  /* Set the data signals (D0-7) of the port to all low (0) */

   outb(192, BASEPORT+3);

   printf("We wrote to the control port");

/*       for (i=0;i<0xFF;i++) {
          outb(i, BASEPORT);
         }
*/

         outb(0x01,BASEPORT+1);
       /* Sleep for a while (100 ms) */
         usleep(100000);

         /* Read from the status port (BASE+1) and display the result */
         printf("status: %d\n", inb(BASEPORT));

         /* We don't need the ports anymore */
         if (ioperm(BASEPORT, 4, 0)) {perror("ioperm"); exit(1);}

         exit(0);
       }

       /* end of example.c */
```

simple.c code

```
/* This program will perform basic io operations using our contructed isa
   isa card. It will write a two bytes and then read them to see if the
   isa card works.

   Authors: Matt Gilbert and Chris Gee
   Date: 4/23/98

*/

#include <asm/io.h>
#include <stdio.h>
#include <unistd.h>
```

```
#define BASEPORT 0x280    /* This is the io address of our isa card */

int main(void){

        int i;   /* used as a counter */
        int k;
        unsigned char loop=0x00;
        int correct=0;
        int error=0;
        int total=0;
        float per_correct, per_error;
        unsigned char first_b = 0xbb; /*Values to be read from the input*/
        unsigned char second_b = 0xbb;


        /* Now we will request the port from the kernel */
        if(ioperm(BASEPORT,4,1)){
                perror("ioperm failed"); /* If ioperm fails to get the */
                return(1);                /* port then we exit */
        }

        outb(192, BASEPORT+3);    /* This sets up the 8255 to have */
                                  /* Port A in mode 2. port B is mode 0 */
                                  /* and port C to be a control and status */
                                  /* for port A */

for(loop=0;loop<=0xff;loop++){
        for(k=0;k<1000;k++){

// while(1){
        /* Next we are going to start the steps to write to output to the
                8255 to be sent on the manchester chip */

        /* going to select the first output latch by selecting it from
           port B */

        outb(0x00,BASEPORT+1);

        /* Next we write some data to the Latch */
        outb(loop,BASEPORT);

        /* Now select the next output latch */
        outb(0x01,BASEPORT+1);

        /* Write more data */
        outb(loop,BASEPORT);

        /* Now latch all the data and go to the default state */
        outb(0x07,BASEPORT+1);

        /* Now we set the counters by putting the enable bit high */
        outb(0x0F,BASEPORT+1);
```

```
        /* Now we set the PL bit to start transmission */
        outb(0x1F,BASEPORT+1);

        usleep(10);

        /* Now reset the PL bit */
        outb(0x0F,BASEPORT+1);

        /* Now we start the read process */

        /* For this all we do is read the values from the latches */


        /* Select the fi+rst latch */
        outb(0x0A,BASEPORT+1);
        /* Read the Data */
        first_b = inb(BASEPORT);
        printf("%x \n",first_b);

        /* Select the second latch */
        outb(0x0B,BASEPORT+1);
        /*Read the Data */
        second_b = inb(BASEPORT);
        printf("%x \n",second_b);

        /* Now compare to the values that were input */
        if(second_b == loop && first_b == loop){
                        correct+=1;
        }
        else{
                error+=1;
        }
        total+=1;
        }



        printf("Number = %x \n",loop);
        printf("correct = %d \n",correct);
        printf("error = %d \n",error);
        printf("total = %d \n",total);

        correct=0;
        total=0;
        error=0;

}
        /* Now we give up the port */
        if(ioperm(BASEPORT, 4, 0)){
                perror("ioperm failed");
                return 1;
```

```
        }
        return 1;

}
```

Appendix B: Actual results from testing test link interface

Sent Data Byte	Number Correct	Number Incorrect	Total
0	8358	1641	10000
1	8260	1740	10000
2	8224	1776	10000
3	8226	1774	10000
4	8305	1695	10000
5	8215	1785	10000
6	8288	1712	10000
7	8309	1691	10000
8	8269	1731	10000
9	8296	1704	10000
A	8342	1658	10000
B	8313	1687	10000
C	8345	1655	10000
D	8336	1664	10000
E	8292	1708	10000
F	8353	1647	10000
10	8321	1679	10000
11	8363	1637	10000
12	8354	1646	10000
13	8307	1693	10000
14	8325	1675	10000
15	8361	1639	10000
16	8319	1681	10000
17	8387	1613	10000
18	8316	1684	10000
19	8274	1726	10000
1a	8299	1701	10000
1b	8313	1687	10000
1c	8332	1668	10000
1d	8343	1657	10000
1e	8275	1725	10000
1f	8345	1655	10000
20	8303	1697	10000
21	8324	1676	10000
22	8305	1695	10000
23	8338	1662	10000
24	8284	1716	10000
25	8301	1699	10000
26	8298	1702	10000
27	8276	1724	10000
28	8245	1755	10000
29	8383	1617	10000
2a	8386	1614	10000

2b	8367	1633	10000
2c	8256	1744	10000
2d	8407	1593	10000
2e	8331	1669	10000
2f	8438	1562	10000
30	8282	1718	10000
31	8334	1666	10000
32	8261	1739	10000
33	8390	1610	10000
34	8247	1753	10000
35	8268	1732	10000
36	8284	1716	10000
37	8325	1675	10000
38	8367	1633	10000
39	8433	1567	10000
3a	8335	1665	10000
3b	8383	1617	10000
3c	8334	1666	10000
3d	8350	1650	10000
3e	8347	1653	10000
3f	8303	1697	10000
40	8209	1791	10000
41	8278	1722	10000
42	8284	1716	10000
43	8270	1730	10000
44	8299	1701	10000
45	8392	1608	10000
46	8438	1562	10000
47	8442	1558	10000
48	8205	1795	10000
49	8275	1725	10000
4a	8299	1701	10000
4b	8326	1674	10000
4c	8266	1734	10000
4d	8410	1590	10000
4e	8311	1689	10000
4f	8387	1613	10000
50	8299	1701	10000
51	8345	1655	10000
52	8307	1693	10000
53	8247	1753	10000
54	8342	1658	10000
55	8301	1699	10000
56	8265	1735	10000
57	8330	1670	10000
58	8320	1680	10000
59	8298	1702	10000
5a	8289	1711	10000

5b	8338	1662	10000
5c	8291	1709	10000
5d	8242	1758	10000
5e	8301	1699	10000
5f	8366	1634	10000
60	8380	1620	10000
61	8291	1709	10000
62	8370	1630	10000
63	8319	1681	10000
64	8289	1711	10000
65	8261	1739	10000
66	8340	1660	10000
67	8274	1726	10000
68	8403	1597	10000
69	8328	1672	10000
6a	8359	1641	10000
6b	8341	1659	10000
6c	8330	1670	10000
6d	8288	1712	10000
6e	8358	1642	10000
6f	8391	1609	10000
70	8322	1678	10000
71	8367	1633	10000
72	8404	1596	10000
73	8307	1693	10000
74	8353	1647	10000
75	8333	1667	10000
76	8282	1718	10000
77	8211	1789	10000
78	8371	1629	10000
79	8339	1661	10000
7a	8296	1704	10000
7b	8306	1694	10000
7c	8338	1662	10000
7d	8263	1737	10000
7e	8276	1724	10000
7f	8237	1763	10000
80	8419	1581	10000
81	8224	1776	10000
82	8351	1649	10000
83	8221	1779	10000
84	8354	1646	10000
85	8306	1694	10000
86	8279	1721	10000
87	8233	1767	10000
88	8308	1692	10000
89	8358	1642	10000
8a	8305	1695	10000

8b	8350	1650	10000
8c	8370	1630	10000
8d	8245	1755	10000
8e	8219	1781	10000
8f	8258	1742	10000
90	8238	1762	10000
91	8272	1728	10000
92	8261	1739	10000
93	8199	1801	10000
94	8356	1644	10000
95	8394	1606	10000
96	8262	1738	10000
97	8183	1817	10000
98	8403	1597	10000
99	8304	1696	10000
9a	8339	1661	10000
9b	8202	1798	10000
9c	8346	1654	10000
9d	8240	1760	10000
9e	8276	1724	10000
9f	8321	1679	10000
a0	8255	1745	10000
a1	8198	1802	10000
a2	8257	1743	10000
a3	8089	1911	10000
a4	8293	1707	10000
a5	8223	1777	10000
a6	8415	1585	10000
a7	8185	1815	10000
a8	8320	1680	10000
a9	8309	1691	10000
aa	8284	1716	10000
ab	8296	1704	10000
ac	8308	1692	10000
ad	8262	1738	10000
ae	8321	1679	10000
af	8160	1840	10000
b0	8304	1696	10000
b1	8214	1786	10000
b2	8179	1821	10000
b3	8170	1830	10000
b4	8360	1640	10000
b5	8240	1760	10000
b6	8141	1859	10000
b7	8168	1832	10000
b8	8211	1789	10000
b9	8163	1837	10000
ba	8175	1825	10000

bb	8255	1745	10000
bc	8166	1834	10000
bd	8254	1746	10000
be	8155	1845	10000
bf	8294	1706	10000
c0	8291	1709	10000
c1	8219	1781	10000
c2	8254	1746	10000
c3	8319	1681	10000
c4	8163	1837	10000
c5	8134	1866	10000
c6	8262	1738	10000
c7	8210	1790	10000
c8	8177	1823	10000
c9	8239	1761	10000
ca	8192	1808	10000
cb	8241	1759	10000
cc	8210	1790	10000
cd	8205	1795	10000
ce	8184	1816	10000
cf	8311	1689	10000
d0	8295	1705	10000
d1	8219	1781	10000
d2	8117	1883	10000
d3	8300	1700	10000
d4	8121	1879	10000
d5	8201	1799	10000
d6	8173	1827	10000
d7	8303	1697	10000
d8	8197	1803	10000
d9	8221	1779	10000
da	8183	1817	10000
db	8245	1755	10000
dc	8140	1860	10000
dd	8163	1837	10000
de	8234	1766	10000
df	8240	1760	10000
e0	8202	1798	10000
e1	8332	1668	10000
e2	8131	1869	10000
e3	8305	1695	10000
e4	8173	1827	10000
e5	8164	1836	10000
e6	8198	1802	10000
e7	8369	1631	10000
e8	8188	1812	10000
e9	8191	1809	10000
ea	8150	1850	10000

eb	8300	1700	10000
ec	8163	1837	10000
ed	8261	1739	10000
ee	8235	1765	10000
ef	8273	1727	10000
f0	8178	1822	10000
f1	8263	1737	10000
f2	8236	1764	10000
f3	8342	1658	10000
f4	8182	1818	10000
f5	8189	1811	10000
f6	8260	1740	10000
f7	8313	1687	10000
f8	8267	1733	10000
f9	8309	1691	10000
fa	8182	1818	10000
fb	8412	1588	10000
fc	8238	1762	10000
fd	8235	1765	10000
fe	8183	1817	10000
ff	9733	267	10000

Appendix C: Pictures of hardware