Digital X-ray Signal Transmission

Sasha Sproch, Sarah Strohkorb, Cypress Frankenfeld, and Jessica Bethune

Satatus Report

***Abstract*—The ability to transmit digital signals via X-ray will redefine the current boundaries of speed, distance, fidelity, and precision in wireless communications. The immediate objective of this project is to enable transmission of digital X-ray signals by coupling a high speed digital RS232 communication infrastructure to a preexisting X-ray transceiver. Circuits and software on each end of the transceiver system interface with the RS232 protocol, enabling signal speeds of up to 4800 Baud. In the future, transmissions will be higher power and capable of interplanetary communication at much higher transmission rates.**

I. Introduction

X-RAY communication is analogous to fiber optic transmission on a much larger scale– it enables long-distance, high-speed, robust digital communication, with the implication of promising applications: communications with spacecraft reentering the atmosphere during the hypersonic blackout period, high-speed interplanetary communication, secure short-range communication, etc. Technological advances in X-ray communication could also lead to interstellar X-ray navigation.

Dr. Keith Gendreau, of NASA Goddard Space Flight Center, developed an X-ray transceiver system which he used to transmit audio through analog frequency modulation. Originally, he suggested that our team attempt to transmit digital video over the MXS using a USB webcam, but we concluded that demultiplexing the bidirectional USB cables would be impractical for a summer project. Our revised goal brings us a step toward video transmission: an upgrade of Dr. Gendreau's demo from analog to digital transmission capabilities.

Statement of Work

We will couple a high speed digital RS232 communication infrastructure to the MXS transceiver. Our system will drive a 255 nm UV LED with 20 mA (and the capability of driving 10 parallel LEDs with 200 mA), and will filter and amplify the output of the X-ray transceiver. We will use a chat client to establish a unidirectional X-ray communication link between two computers. To make the link bidirectional, we need only add another MXS transceiver and a duplicate of our LED driver and filtering circuitry.

II. The System

The X-ray transceiver system developed at Goddard Space Flight Center by Dr. Gendreau, and used in our tests, consists of an emitter and detector. The emitter, or Modulated X-ray Source (MXS), emits characteristic X-rays focused in a beam and directed toward the detector, also called the Multi-Channel Analyzer (MCA). The MCA generates a short (10 ns) voltage spike corresponding to each detected X-ray photon. The MCA produces a burst of X-rays for each logic 1, and the resulting bursts of voltage spikes vary significantly in amplitude, causing the signal to noise ratio (SNR) to be very low. The effective output signal of the MCA is low level white noise for a logic 0, and higher amplitude white noise for logic 1, as seen in the oscilloscope trace in Figure 1.

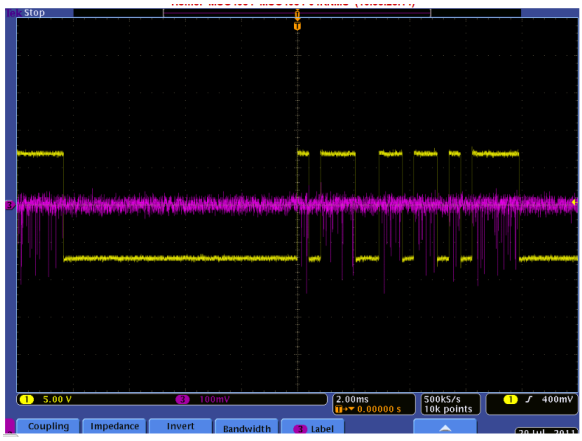


Fig. 1. The yellow waveform is the input signal; A $\pm$ 7.5 V binary representation of two ascii characters ('1' followed by newline) sent with RS232 protocol. The purple waveform is the output of the MCA with no filtering; voltage spikes of about 200 mV can be seen at each space in the input. A negative input voltage (mark) corresponds to low level white noise, and a positive input (space) corresponds to high level white noise.

The irregularity of the individual x-ray pulses makes the signal susceptible to bit errors. Our most challenging task for this project was to filter and condition the output signal into a clean, digital signal, like the input.

Figure 2 depicts a high-level overview of how the MXS works. The ultraviolet (UV) light-emitting diode (LED) shines onto a photocathode, which emits electrons due to the photo-electric effect. The electrons then travel through the electron multiplier (which increases the number of electrons by several orders of magnitude) and are accelerated into a high-voltage target. When hit by the electrons, the electron target emits characteristic X-rays.

We key the LED on and off to modulate the electrons and thus the X-rays. The output of the X-ray detector shows groups of voltage jumps that correspond to the on times for the LED– each space in the input signal.

The physical MXS setup used to test our system is displayed in Figure 3. A high voltage power supply powers the electron target and multiplier, and a signal generator drives the LED. We view the output signal of the detector on an oscilloscope.

To transmit a digital signal using the MXS and give Dr. Gendreau the tools to make a digital transmission demonstration, we manipulate the signal on both the transmitting and receiving ends of the X-ray transceiver system, modeled in

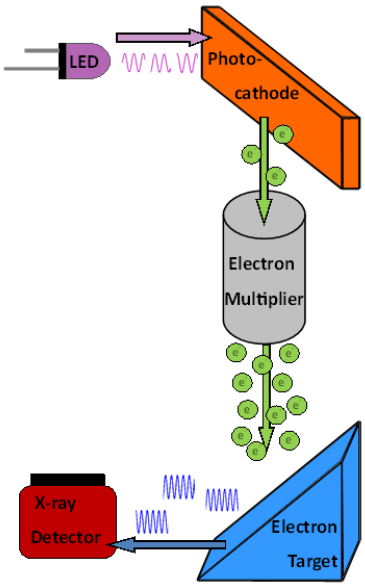


Fig. 2. The MXS transmits X-rays when the UV LED is iluminated. Light hits a photo-cathode, where photon energy is absorbed and released in the form of photo-electrons. These photo-electrons are multiplied and then accelerated into a target, which emits characteristic X-rays to be detected by the photodiode in the MCA.

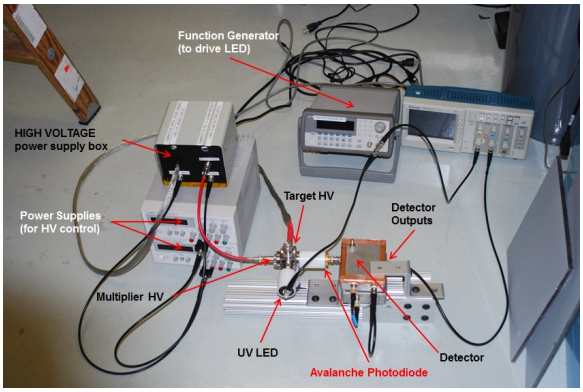


Fig. 3. Lab setup. X-rays are transmitted a distance of less than ten centimeters for safety reasons.

Figure 4. We develop software and hardware on each end to make our system compatible with any RS232 transmission. The software on the transmit end sets the baud rate, serial port permissions, and other settings necessary to send the signal. We pass the signal through a MAX232 integrated circuit (IC) followed by a transistor current switch to ensure that the UV LED receives the proper current and voltage levels. After the signal is sent through the MXS, the output of the detector is weak and noisy due to physical properties of the system beyond our control. We amplify and filter the output of the MCA until it is a close enough approximation to the input signal for the MAX232 chip to transmit it to a computer on the other end without bit errors. Finally, software on the receiving computer ensures that the baud rate and similar settings are compatible, and presents the sent information in an easily understood, visual way.

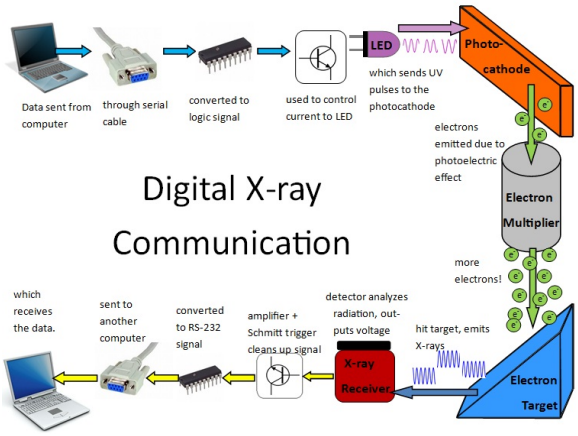


Fig. 4. Abstract model of entire system. A MAX232 and LED control circuit to modulate the LED. A second circuit takes the output signal of the MXS and transforms it back into a digital signal, which is passed through another MAX232 to the second computer.

III. Hardware

The hardware required for this system includes the control circuit used to drive the UV LED which transmits the signal, and the transformer circuit used to clean up the MCA output signal, making it compatible with the receiving computer.

LED Control Circuit

RS232 signals can be sent at voltages ranging from ± 3 to ± 15 Volts. This wide spectrum is dangerous when trying to interface with transistor circuits because the smaller voltages will not drive transistors into saturation, and the larger voltages can create surges fatal to the transistor. The MAX232 solves this problem by transforming the computer's high-amplitude square wave signal into a 0V to 5V TTL signal.

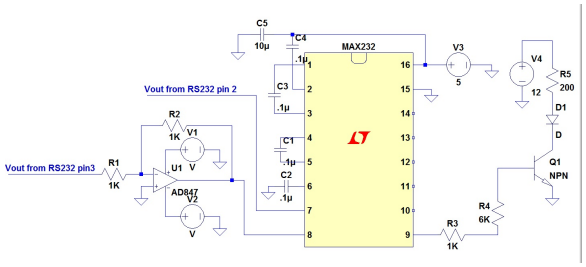


Fig. 5. LED control circuit with an input from the RS232 connector to the LED that controls the MXS. Full page image can be found in the appendix.

The MAX232 does not provide enough current to drive the LED fully on, so we used a transistor switch to modulate power from an external supply. When the input voltage to the amplifier is high, the switch turns on and current can flow from the collector to the emitter of an NPN transistor, and consequently, current can flow through the LED connected to the transistor's collector. When the input to the amplifier is low, the switch turns off, cutting current to the LED.

Transceiver Output Conditioning Components

After transmission through the MXS and MCA, the signal needs to be conditioned into a digital signal, as represented

in Figure 6. A combination of several building blocks are used to construct this filter.

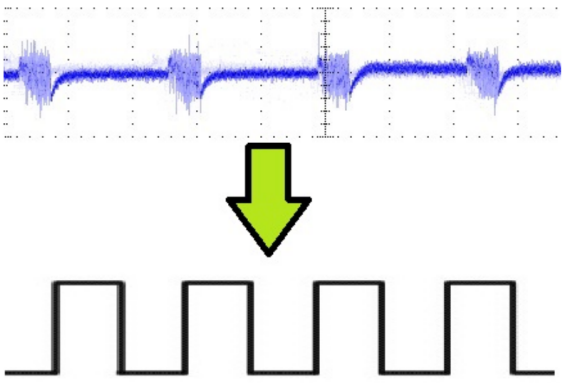


Fig. 6. Top: example of the output of the MXS when given a square wave input. Bottom: desired signal after filtering and conditioning.

Amplifiers: We use simple inverting/non-inverting voltage amplifiers made from either an AD848 or AD847 operational amplifier (op-amp). The AD848/AD847 has a higher gain bandwidth product (GBP) and a higher slew rate than the typical op-amp, such as the TL081. The higher GBP of the AD848 (175 MHz) and AD847 (50 MHz), allows them to function at much higher frequencies than the TL081 (3 MHz GBP). Likewise, the higher slew rate of of the AD848/AD847 (300 V/us) allows a faster response time to voltage changes than that of the TL081 (16 V/us). The extremely high GBP on the AD848 creates limitations, however– it requires a minimum gain of 5, causing voltages in a cascaded circuit to rail easily. Our team therefore uses AD848 op-amps for high gain amplifiers and AD847 op-amps for low gain amplifiers and unity buffers.

Peak-hold Detector: The peak-hold detector, the core of our MXS output circuitry, allows us to transform the noisy pulses from the MXS into a comparatively clean waveform. Our project necessitates a high slew rate in order to detect narrow voltage spikes (on the order of 10 ns).

To keep the AD848's output from saturating at V_{CC}, We adjust the gain and offset of the amplifier directly before the peak-hold detector, tuning it to respond best to the MCA signal. Because we observe highly varied MCA outputs, (corresponding to differences in temperature, spatial positioning, proximity to power supplies, etc.) we do not recommend fixed resistors for these jobs. For simplicity, however, and to limit parasitic resistances/capacitances in our breadboards, we choose fixed gains, and leave only the offsets variable.

The peak-hold detector's behavior is dependent on its RC time constant and op-amp GBP. The RC value determines the drop-off speed after each voltage peak. If held too long, a peak interferes with signal transmission by widening the apparent width of the bits being sent; if the peak decays to quickly, the circuit output drops to 0V in between peaks in the middle of an on bit, creating a false 0. Figures 7 and 8 demonstrate peak-hold detector responses to square waves created with a signal generator at key RC values.

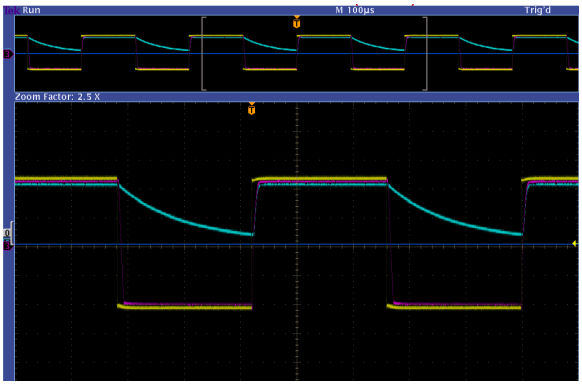


Fig. 7. Peak-hold response to a 5 KHz square wave. The output signal (blue) falls to zero just as the next peak retriggers it.

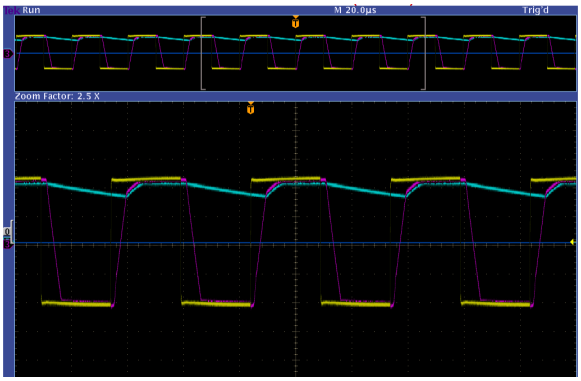


Fig. 8. Peak-hold response to a 50 KHz square wave. The blue output signal never falls to zero.

Schmitt Trigger / Comparator: We use a Schmitt trigger to make the output of the peak hold detector a clear digital signal. A Schmitt trigger has two threshold levels; as long as the signal is below the lower threshold, the Schmitt trigger outputs 0V, but once the signal is above the higher threshold, the output is 5V. While the signal is between the two thresholds, the output of the Schmitt trigger does not change.

The IC Schmitt trigger (74HC14) only takes input signals in the voltage range of 0V to 5V, so we create our own Schmitt trigger using an op-amp with an adjustable virtual ground and a variable gain. This enables us to tune our Schmitt trigger to give us the level of hysteresis we need. Keeping these values adjustable is advisable because the MCA output is highly varied.

MAX232: Once the signal is digital, the MAX232 IC can convert it from 0 to 5 volts into a $\pm$7.5V signal or vice versa. The MAX232 offers very robust signal conditioning for RS232 drive capability.

Transceiver Output Conditioning Circuit

Our final conditioning circuit, seen in Figure **??**, utilizes a combination of the aforementioned building blocks. First,

an AD848 conditioning amplifier, with noise-filtering capacitors near the pins, increases the resolution and SNR of the MCA's output. Second, an amplifier with an adjustable offset centers the signal around zero. Ideally, the gain would also be adjustable, but for testing purposes we choose fixed gains. This second amplifier is non-inverting for the positive MCA channel, and inverting for the negative channel. We use only the negative channel due to its stronger signal. Because the signals are just inversions of each other, using both adds nothing to the system. When adjusting the offset, we want to eliminate false 1s, but allow false 0s. Output from the gain/offset amplifier travels into a peak-hold detector (made of a 1N4148 diode and an RC filter), which leads into our discrete-component Schmitt trigger. At this point, the signal is digital with some false 0s. We run the signal through a MAX232, another peak-hold detector, and an AD847 unity-gain buffer. We then pipe the final signal, seen in Figure **??**, into the reciever pin (pin 2) of an RS232 cable.

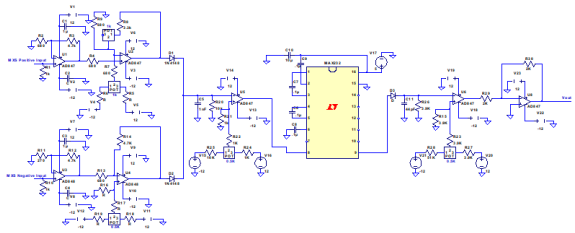


Fig. 9. Conditioning circuit, tuned roughly for 4800 baud transmission. Full page image can be found in the appendix.

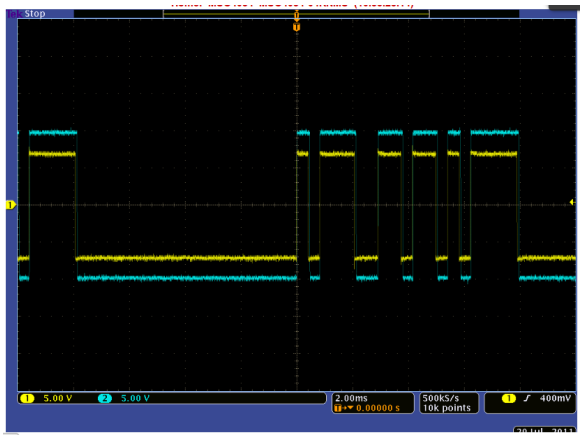


Fig. 10. Successful transmission: the filtered output of the MCA (blue) is an amplified copy of the MXS input (yellow).

Troubleshooting

We encountered the following defects in the MXS output signal; they may be endemic to our specific experiment (our setup, or local radio interference) or to the system as a whole. The signal quality degrades toward the end of the pulse at low frequencies, rendering a low pass filter useless. The occasional and unpredictable scarcity of high amplitude peaks during one LED pulse makes a peak-hold detecor difficult– if possible– to properly tune. Something in the MCA seems to have a thermal sensitivity, because extended use corresponds with severe signal degredation which resolves itself if the MCA is unplugged and otherwise left alone for an hour or more. Additionally, target high voltage source (THVS) malfunctions have skewed some results during testing. In its upright position, the THVS tends to superimpose a 1 kHz sine wave over the output, as seen in Figure **??**. Lose components or a grounding problem in the circuitry may be causing this phenomenon. As a temporary solution we turn the THVS box on its side until it can be repaired or replaced.

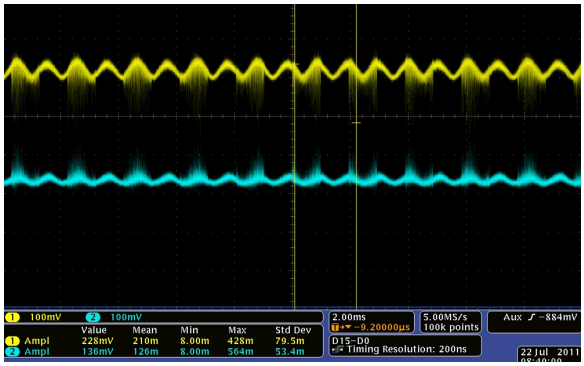


Fig. 11. A 1 kHz sine wave can be seen on both the positive (yellow) and negative (blue) MCA outputs. The input to the MXS is just under a 1 kHz a square wave– the input function appears to have no effect on the superimposed sine wave.

IV. Software

Custom Signals

Modern high-speed digital communication protocols, because of elaborate error control and handshaking, can impede our ability to control and debug the signals at the bit-level.

Fig. 12. Three pins connecting the microcontrollers.

To establish complete signal control, we create a simple digital communication protocol by programming two microcontrollers. We use three wires to connect them, which we refer to as *Mark*, *Space*, and *Confirm*.

This protocol is unidirectional. A single bit is sent in 5 stages.

1) Microcontroller A raises voltage on the mark pin.
2) Microcontroller B responds by raising the confirm pin.
3) A sees the confirm pin has been raised, and responds by lowering the mark pin.
4) B sees that the voltage has been lowered once again, and responds by lowering the confirm pin.
5) A sees that the confirm pin has been lowered, and thus decides it is safe to send another bit (repeat from beginning).

The same process can be used to send a space bit. Mark and space bits correspond to 1s and 0s in a digital signal. The system functions irrespective of transmission speed; microcontroller A will wait indefinitely for the confirmation signal before it proceeds. This circumvents any problems arising from signal delay.

UART Between Microcontrollers

We use the built in Universal Asynchronous Receiver/Transmitter (UART) in the microcontrollers to send serial data. Hooking the transmit pin of one microcontroller to the receive pin of another, we send bytes of data at a specified baud rate using the `Serial.write()` and `Serial.read()` functions. This approach is a proof of concept: we can demonstrate transmission of data in the RS232 protocol via a microcontroller.

Microcontroller To Computer

To achieve a connection between the microcontroller and the computer, the low output voltages of the microcontroller need to be inverted and amplified into the RS232 range with a MAX232 chip.

A Unix based system developed in 1962, RS232 protocol uses the `stty` command to establish baud rate, end of a line or file characters, etc. We create a Bash program that establishes base settings uniform across devices, and use a Unix shell to receive data from the microcontroller by running `cat /dev/ttyS0`, which outputs the characters from the serial port into the shell display.

Computer Loopback

To ensure that the computer can send a signal, we test with an RS232 loopback connection– we interconnect pins 1, 6, and 4, pins 7 and 8, and pins 2 and 3 (the receive and transmit pins). We then run `echo A > /dev/ttyS0` while simultaneously waiting for a response with the command `cat /dev/ttyS0`. The transmission sent by the transmit computer is read by the same computer and output in a terminal. This allows us to easily test serial port settings and maintain compatability.

RS232 Between Computers

We connect two computers using the above loopback connection (with the exception that pins 2 and 3 are interconnected between computers with a null modem), ensure that all the `stty` settings are the same, and then run `echo A > /dev/ttyS0` in one computer (transmit) while running `cat /dev/ttyS0` in another (receive). The transmit computer sends the signal and the receive computer prints the signal to its terminal. This demonstrates that the two computers are able to transmit and receive RS232 data between each other.

Chat Client

A short Bash script automates the communication between two computers. It automatically maintains consistent settings and listens for an input signal between devices. The commands `read input; echo "${input}" > /dev/ttyUSB0` simplify the user interface by enabling users to directly input multiple messages while concurrently receiving data from the linked computer, without the need for manual port specification.

Remote Terminal

We use a getty server running as an upstart service on the host computer. We connect, via RS232, from another computer running minicom. In order to connect properly, we need to configure getty correctly, and configure minicom to match the getty. By connecting via a remote terminal, we have full control over one computer via a UNIX Shell in another computer, all running through RS232.

File Transfer

Once the remote terminal is set up, we use the ZMODEM protocol to transfer files from one computer to another through minicom. File transfer speeds are fast enough to transfer an image, but too slow to transfer a 70 MB file within a reasonable amount of time (24 hours).

V. Results

Proof of Concept

The LED and photodiode proof of concept shown in Figure **??** represents the ideal transceiver behavior, with perfect MXS modulation and MCA detection of the X-ray signals. We put the RS232 transmit computer signal through a MAX232 chip to drive the LED, which is aimed at a photodiode that can detect every pulse (marked by a voltage increase). This signal goes through two inverting amplifiers so that the signal ranges roughly from 0-5V. Another MAX2332 transforms the signal into RS232 voltage levels, which are sent back to the receiving computer. Using this setup, we are able to use the chat client to transmit digital signals at 115,200 baud without any errors.

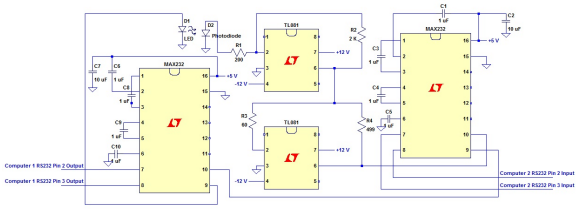


Fig. 13. Proof of concept. An LED and photodiode replace the X-ray transceiver. The RS232 signal, passing first through a MAX232, drives the LED. The light signal is detected by the photodiode, then amplified and piped through a second MAX232 so that the recieve computer can read it via RS232 protocol. Full page image can be found in the appendix.

The LED must be in a very dark environment when in use in order to ensure that the detected signal reaches 0V when the LED is off. One large difference between this proof

of concept and our actual setup is that the output of the photodiode requires no filtering. When it detects LED light pulses, it outputs a signal with a high enough resolution to be piped, amplified but unfiltered, into to the MAX232 and then through the computer without perceivable errors. The X-ray detector output, on the other hand, could be best characterized as high and low amplitude noise, with a highly variable SNR, requiring significant filtering.

RS232 Transmission

July 28, 2011 marks the first successful digital transmission via X-rays. We use RS232 protocol and the simple Bash `echo` command in a terminal shown in Figure **??**. The text of the first signal, sent at 300 baud, is a quote by Orville Wright: "Isnt it astonishing that all these secrets have been preserved for so many years just so we could discover them!" Following the first successful transmission, we incrementally increase the baud rate, checking for frequency of errors and adjusting the discrete component values as necessary. Concurrently, we test the chat program and file transmission at these increasing baud rates. We find that a signal with an approximate error rate of one false bit every few seconds can be transmitted at speeds up to 4800 baud, after which the signal begins to degrade. Lower frequencies have a smaller error rate, and any transmission speed above 4800 baud has very frequent errors.

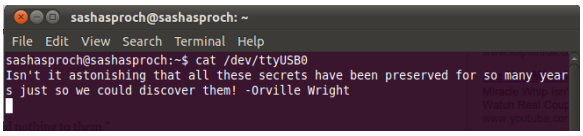


Fig. 14. Screenshot of the first digital X-ray transmission. Sent via RS232 protocol with an MXS/MCA transceiver system.

VI. Conclusion

Using the X-ray transceiver system, we successfully transmit digital data via an X-ray signal. The key hardware components of our project are the LED driver used to modulate the UV LED in the MXS, and the filtering circuitry which conditions the output of the MCA. The current setup can transmit clean digital signals at 4800 baud; increasing the baud rate introduces false bits into the data stream.

VII. Future Work

Hardware

Our finished circuit successfully transmitted the first digital X-ray signal on July 28, the day before the end of the internship. Consequently, there is a lot of opportunity for tuning and improvements. Had we more time, our immediate next steps would be to re-arrange the circuit. For improvements, we recommend the following changes: keep the first three amplifiers, but move the MAX232 after the fourth amplifier (U6 in the Figure **??**) and, in place of the final amplifier (U7), feed the signal back through pin 10 of the MAX232 and let pin 7 be the output that feeds directly into the computer. Then the reference of R26 can be the -12V rail instead of ground. These changes will likely improve signal quality because the Schmitt trigger made with U6 will have a rail-to rail signal, enabling more hysteresis. Also, feeding the output of the op-amp to the computer is much more dangerous than doing likewise with the MAX232s output, because the MAX232 conditions the output specifically for RS232 drive capability. The next step would be to perfect the resistor and capacitor values throughout the whole circuit, specifically in the peak-hold detectors and the amplifier gains.

The circuit in the diagram in Figure **??** is currently tuned (to a very rough approximation) to a transmission speed of 4800 baud. A Monte Carlo analysis could help optimize the gain and offset parameters for various baud rates.

Within the transceiver system, there are improvements that can be made to increase the fidelity of transmissions. Another NASA team is making an electron focusing lens to integrate with the MXS in order to improve the strength, precision, and SNR of the signal. For a much more reliable and consistent output over a much wider range of frequencies, we suggest reworking the MCA circuit from the ground up with the idea of digital transmission in mind. This should enable the detector circuit to process digital signals of much higher bit rates. If high bit rates are successfully sent, the next step would be to transmit signals via different, faster protocols such as Ethernet. Two transceiver systems would be useful for testing bidirectional transmissions.

Software

To improve the signal quality on the software front, compression, encryption, and error correction could be used. Future software work can also be done to improve the quality of the transceiver demonstration, making it more user-friendly. It would be useful to implement a program to stream video over RS232 or Ethernet, depending on what hardware has been completed, and then use this to stream a video, interrupting the signal partway through and demonstrating that the video has been halted. An easy-to-use RS232 file transfer system and chat client, one which doesn't require UNIX knowledge or experience using a terminal, would make the current demonstration more widely accessable.

Acknowledgements

We would like to thank Dr. Brad Minch, Dr. Jos Oscar Mur-Miranda, and Dr. Siddhartan Govindasamy for their assistance and advice with the circuitry on this project, and we would like to thank Dr. Keith Gendreau and Dr. Steve Holt for their mentorship. We would also like to thank Amber Clark for her help editing this report.

Appendix

A. Complete Software Guide

A complete software guide can be found on nasa.olin.edu/projects/2011/dcom/software_guide.php

B. Arduino Custom Transmitter

```
// Initialize instance variables
int highpin = 13;
int lowpin = 12;
int commpin = 11;
byte test = B10101010;
void setup()  {
// tell each pin what it does (input or output)
pinMode(highpin, OUTPUT);
pinMode(lowpin, OUTPUT);
pinMode(commpin, INPUT);
Serial.begin(9600);
send(test);
}

void loop() {
if (Serial.available() > 0) {
     // read the incoming byte:
     test = Serial.read();
send(test);
     // say what you got:
    }

}

// tells the arduino to send each bit
// of the byte that you give it
// individually
boolean send(byte b){
Serial.print("sending: ");
Serial.println(b,BIN);
// use a bitwise "and" to compare
// each byte to a byte with only
// one bit. Then decide to send
// a "high" if the bit is 1, and a
// "low" if the bit is 0
if ((b & B10000000) == B10000000){

high();Serial.println(1);}
else{low();Serial.println(0);}
if ((b & B01000000) == B01000000){

high();Serial.println(1);}
else{low();Serial.println(0);}
if ((b & B00100000) == B00100000){

high();Serial.println(1);}
else{low();Serial.println(0);}
if ((b & B00010000) == B00010000){

high();Serial.println(1);}
else{low();Serial.println(0);}
if ((b & B00001000) == B00001000){

high();Serial.println(1);}
else{low();Serial.println(0);}SSSS
if ((b & B00000100) == B00000100){
high();Serial.println(1);}
else{low();Serial.println(0);}
if ((b & B00000010) == B00000010){

high();Serial.println(1);}
else{low();Serial.println(0);}
if ((b & B00000001) == B00000001){
high();Serial.println(1);}
else{low();Serial.println(0);}
return true;
}

// send a "high" bit
// returns true when done
boolean high(){
// raises the Mark pin ("High") to 5 V
digitalWrite(highpin,HIGH);
// waits until the Com pin is high
while (digitalRead(commpin) == LOW){
}
// turns off the Mark pin
digitalWrite(highpin,LOW);
// waits until the Com pin is low
// before finishing
while (digitalRead(commpin) == HIGH){
}
return true;
}

// sends a 0 bit (uses same process as
// high() does to send a 1)
boolean low(){
digitalWrite(lowpin,HIGH);
while (digitalRead(commpin) == LOW){
}
digitalWrite(lowpin,LOW);
while (digitalRead(commpin) == HIGH){
}
return true;
}
```

C. Arduino Custom Receiver

```
int highpin = 13;
int lowpin = 12;
int commpin = 11;

void setup()  {
pinMode(highpin, INPUT);
pinMode(lowpin, INPUT);
pinMode(commpin, OUTPUT);
Serial.begin(9600);
}

void loop() {
// if the Mark pin is on, it's
```

```
// receiving a 1
if(digitalRead(highpin) == HIGH){
// do the handshake
digitalWrite(commpin,HIGH);

// wait until the Mark pin turns off
// again, meaning the other computer
// has received the handshake
while(digitalRead(highpin) == HIGH){
}
// then turn off the Com pin
digitalWrite(commpin,LOW);
// output the bit you received
Serial.print(1);
}

// if the Space pin is on it's
// receiving a 0
if(digitalRead(lowpin) == HIGH){
// do the handshake
digitalWrite(commpin,HIGH);

// wait until the Space pin turns off
// again, meaning the other computer
// has received the handshake
while(digitalRead(lowpin) == HIGH){
}
// then turn off the Com pin,
// completing the handshake
digitalWrite(commpin,LOW);
// output the bit you received
Serial.print(0);
}
}
```

D. Computer to Computer Chat Script

```
#!/bin/bash

port="/dev/ttyUSB0"

stty speed 4800 rows 0 columns 0 line 0
intr ^C quit ^\\ erase ^? kill ^U eof ^A
start ^Q stop ^S susp ^Z rprnt ^R werase
^W lnext ^V flush ^O min 1 time 0 -parenb
-parodd cs8 hupcl -cstopb cread -clocal
-crtscts -ignbrk -brkint -ignpar -parmrk
-inpck -istrip -inlcr -igncr -icrnl -ixon
-ixoff -iuclc -ixany -imaxbel -iutf8 -opost
-olcuc -ocrnl -onlcr -onocr -onlret -ofill
-ofdel nl0 cr0 tab0 bs0 vt0 ff0 -isig
-icanon -iexten -echo -echoe -echok -echonl
-noflsh -xcase -tostop -echoprt -echoctl
-echoke < ${port};
stty speed 4800 < ${port}

clear;
bold=`tput smso`;
offbold=`tput rmso`;
trap 'kill $!; pkill cat;exit' INT

while true
do
cat ${port}
done &


while true
do
read a
echo "${bold}${a}${offbold}" > ${port}
done

kill $!;pkill cat;
exit
```

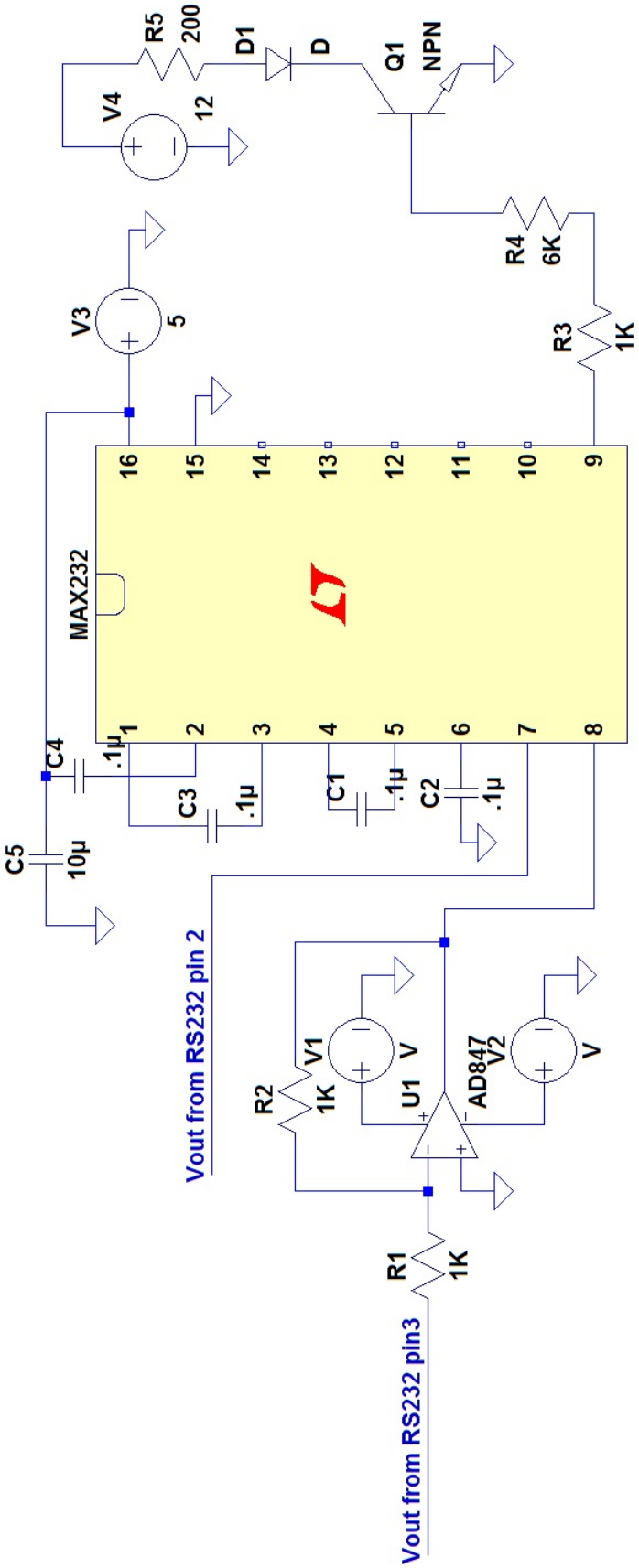
E. Arduino to Arduino Transmitter

```
#include <SoftwareSerial.h>

#define rxPin 2
#define txPin 3

SoftwareSerial mySerial =
SoftwareSerial(rxPin, txPin);

void setup()  {
// setup software serial for output
pinMode(rxPin, INPUT);
pinMode(txPin, OUTPUT);
mySerial.begin(9600);
}

void loop() {
mySerial.print(2,BIN);
delay(500);
}
```

F. Arduino to Arduino Receiver

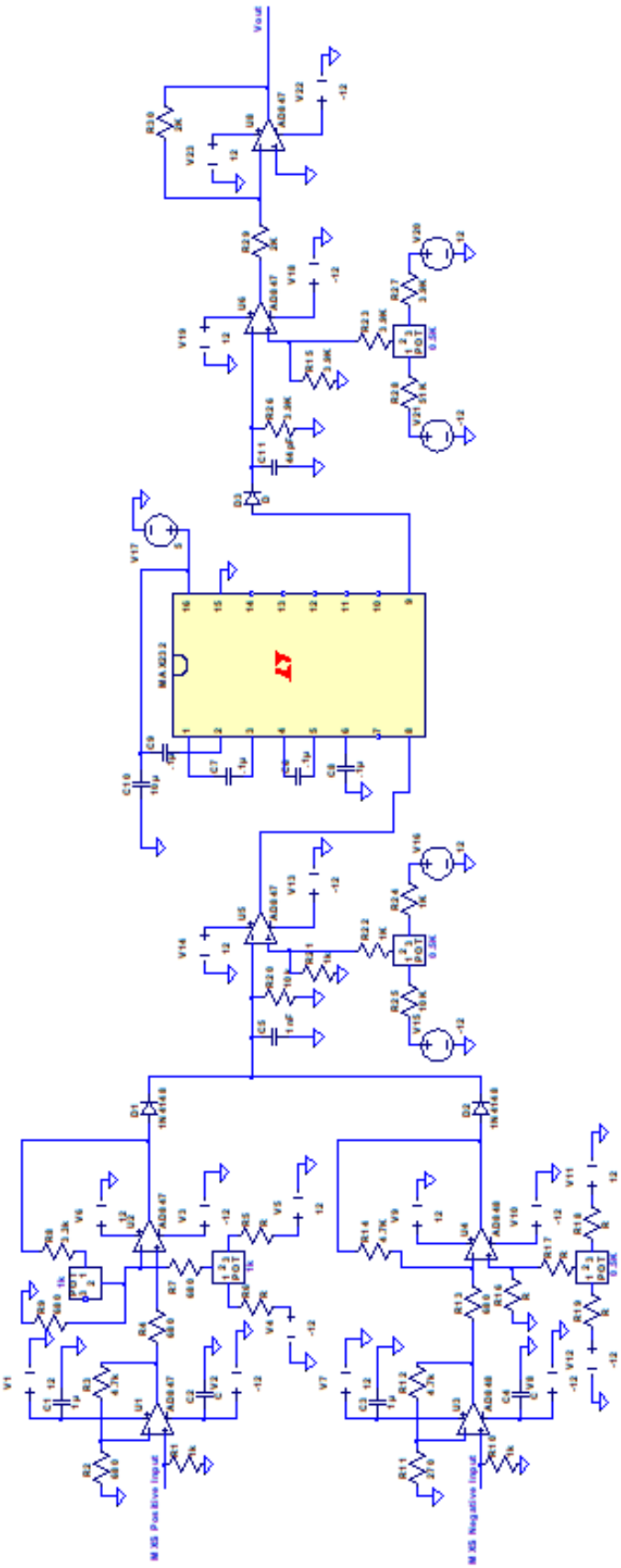
```
#include <SoftwareSerial.h>

#define rxPin 2
#define txPin 3

SoftwareSerial mySerial =
SoftwareSerial(rxPin, txPin);

void setup()  {
// setup software serial for input
pinMode(rxPin, INPUT);
pinMode(txPin, OUTPUT);
mySerial.begin(9600);

// setup hardware serial port
Serial.begin(9600);
}
```

```
void loop() {
char someChar = mySerial.read();
Serial.print(someChar,BIN);
delay(500);
}
```



G. LED Control Circuit

H. Conditioning Circuit



I. Proof of Concept