Programmable Theater Lighting Controller

Kris Kopel and Jeff Sand

Electrical Engineering

Advisor: Dr. Donald Schertz

May 16, 2001

Abstract

The project is a console for theater lighting control. It interfaces with a personal computer (PC) using the Universal Serial Bus (USB) giving a friendly graphical user interface for lighting control. The console, based upon the Motorola 68376 microcontroller, controls lighting instruments through the use of the industry standard DMX512 lighting control protocol. Up to 16 sets of 8 external faders and momentary push buttons can be added for real time lighting control on the console. The input blocks use the queued serial peripheral interface (QSPI) and are designed to be modular so the system can be configured with different amounts and types of user inputs. The prototype system consists of two blocks of 8 faders and 8 buttons each as well as a third block containing three master fader controls. The system outputs the dimmer control data in DMX512 format using the serial communication interface (SCI) and an RS-485 transmitter.

Also, a USB peripheral interface, firmware to control it, and a windows driver to access it, was designed for future department use.

Table of Contents

Introduction

The project goal was to build a self-contained console for theater lighting control that will plug into the USB port on an x86 computer. It allows control of lighting instruments through the use of the industry standard DMX512 lighting control protocol. The system consists of a PC connected through the Universal Serial Bus (USB) to a receiver chip and a microcontroller that will then interface to user input hardware and output circuitry. The microcontroller is a Motorola 68376. Also part of the project is a user input interface for a set (or multiple sets) of faders and buttons for real-time control of the lighting instruments connected to the console. The input blocks are designed to be modular so the system can be configured with different amounts and types of user inputs. The system outputs the dimmer control data in DMX512 format using a UART and an RS-485 receiver. The PC is used to display the status of the console and to let the user program cues into memory.

DMX512 Lighting Control

Modern theatrical and concert lighting systems, whether in a permanent installation or a touring application, typically use low-voltage signals to control the levels of various dimmers in the venue for stage lighting. DMX512 is the most popular industry standard digital communication protocol for theatrical, concert, and special event lighting equipment. The protocol was designed in response to a number of competing proprietary systems developed in the early and mid 1980's. By 1990, DMX512 had virtually replaced all of these protocols and had become the de facto standard. It uses an asynchronous digital bitstream operating at 250kbps with RS-485 as the physical implementation. It is a fairly simple yet extremely versatile communication scheme. DMX512 has also been used to operate fog machines and various other special effects equipment in addition to lights. There are some packages on the market for turning a PC into a lighting controller; however, these usually cost $1000 or more. One goal of this project is to design a similar system at significantly lower cost.

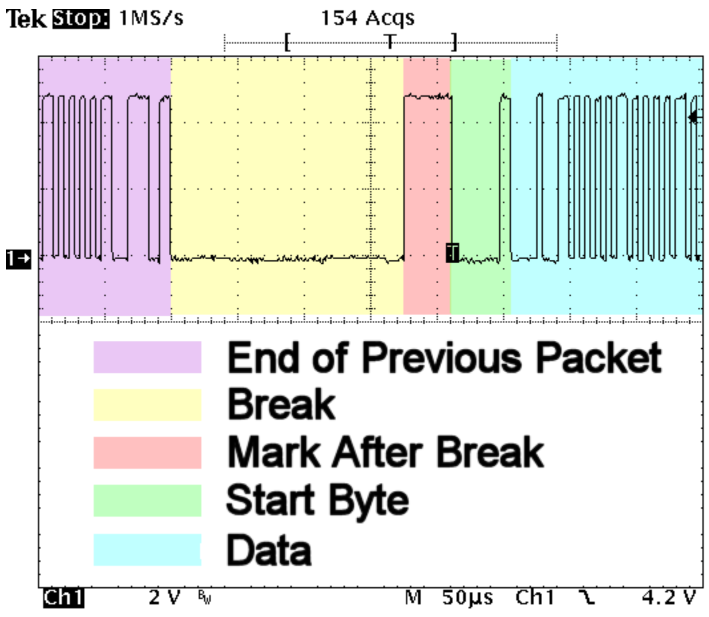

Figure 1 – DMX Data Stream

The image to the left shows the format of the DMX512 stream. This is an actual scope trace of the system output.

Break - Signals the beginning of a new DMX packet. Minimum 22 low bits (88us), shown here as 176us. Varies up to ~260us.

Mark After Break (MAB) – High pulse after break. Minimum 2 bits (8us), system output is 36us.

Start Byte – First data packet, identifies stream as lighting dimmer levels. A data frame with value 0x00.

Data – Dimmers Values 1-512. The frames are in the format of one stop bit, 8 data bits from least significant to most significant, and two stop bits. The DMX specification allows the line to idle high between frames for up to one second, but this system does not require any idle time between frames to allow for near theoretical maximum refresh rates.

The Universal Serial Bus

The Universal Serial Bus is a newer PC peripheral interface that offers several advantages over the ISA bus or the parallel port. First and foremost, it is a faster interface than the parallel port. This is critical for this application since data will have to move very fast for real-time control of lighting equipment. Second, the USB allows the system to be connected to the PC without needing to open the box or configure any internal hardware. This simplifies installation of a new product. Third, there are USB receiver chips on the market that offer a simple way to communicate with the USB bus with minimal external hardware. Finally, using the USB provides the opportunity to learn about a new PC communication method which can be used in future Bradley University Department of Electrical Engineering projects.

System Description and Hardware Block Diagrams

Functional Description – Inputs and Outputs

The overall system block diagram is shown in Figure 2. The inputs to the system are a PC with keyboard and mouse and a control console (also housing the microcontroller) with a series of faders and buttons. The microcontroller and the analog inputs are all contained in one console, which will sit in front of the user next to the PC. The PC is used for a graphical user interface with programming capabilities. The faders and buttons are mounted on the console and will be used for real-time analog control of lights during a show and for programming cues. The user will be able to control the system using the analog controls, pre-programmed cues and sequences on the computer, or a combination of both. As shown in the block diagram in Figure 3, the faders and buttons can be in blocks of 8 each. These blocks are designed such that they are modular, which will allow the console to be built with however many fader blocks the customer wishes to order. The outputs from the system are the PC monitor and the DMX512 output. The PC monitor is used to display the status of the system and the DMX512 signal is the differential lighting control output. The DMX512 output feeds into a 5-pin XLR connector to plug into DMX devices.

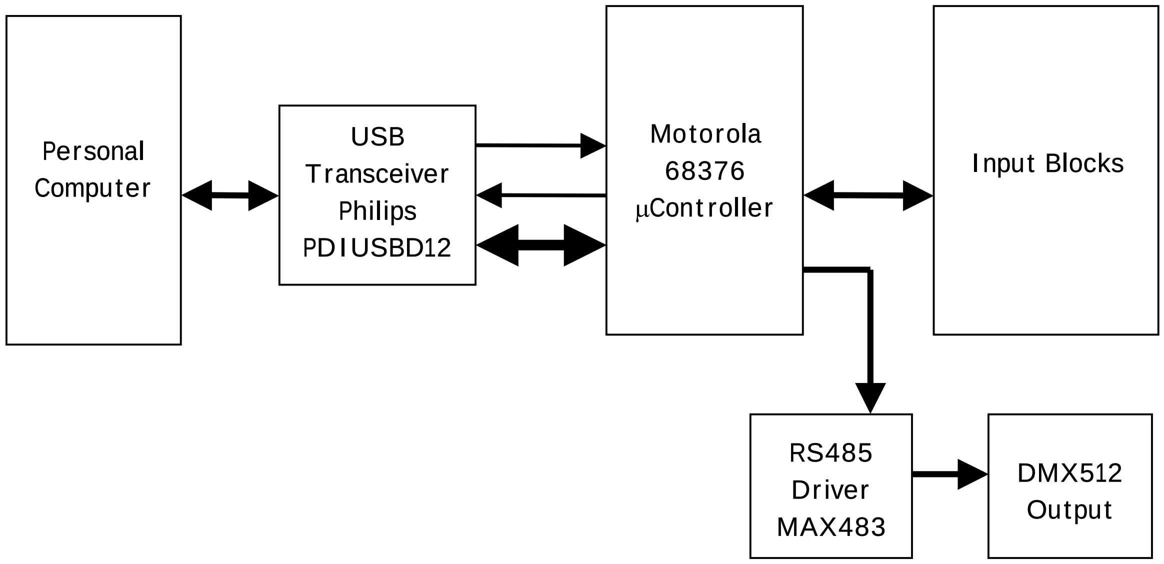

Figure 2 – Complete System Block Diagram

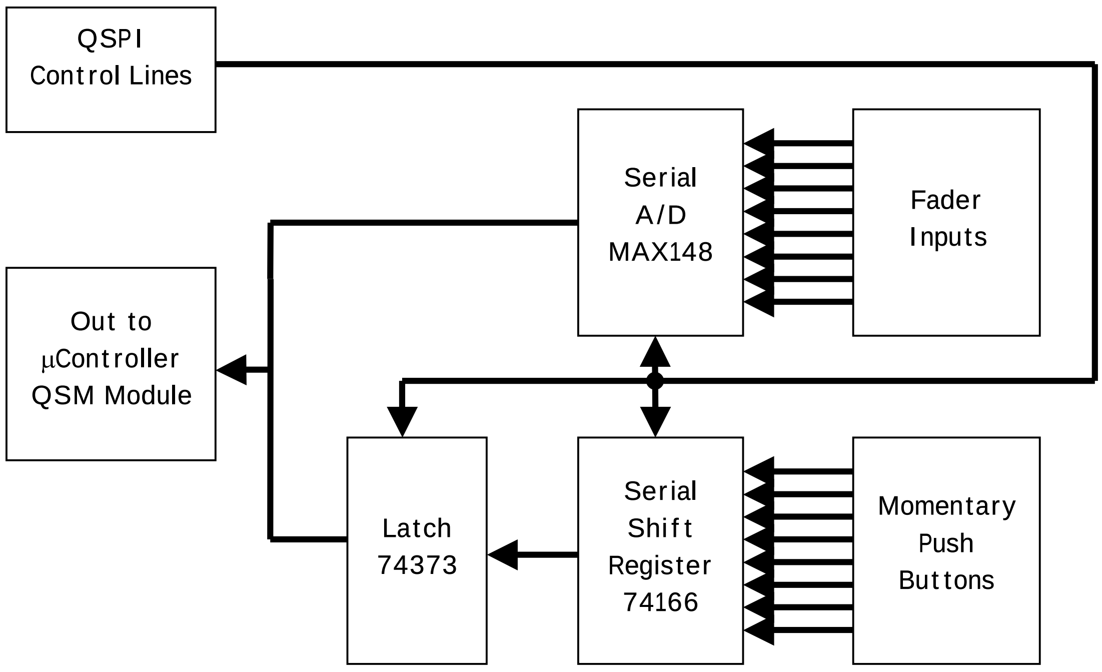

Figure 3 – Input Block

Functional Description – Inner Workings

There are three main functional blocks in the system. The first is the PC that handles all programmable aspects of the controller. It has a GUI for programming and displaying the status of the system. It is connected to the microcontroller block using a USB connection.

The second block is the microcontroller, a Motorola 68376. It is connected to a USB transceiver for communication with the PC and output to the DMX512 interface. It has serial input from the input blocks. The microcontroller's serial communication interface (SCI) handles the output in DMX512 format. The output of the SCI is connected to a MAX483 RS-485 transmitter.

The third system component is the set of input blocks. As this is a modular design, each input block contains a set of faders and push buttons which send a single serial data line to the microcontroller. This is accomplished using the Queued Serial Peripheral Interface (QSPI) on the 68376. The A/D used is a MAX148 8-channel 10-bit SPI-compatible chip. The buttons are connected to a 74166 serial output shift register, which in turn is connected to a latch connected to the QSPI data input. This allows the buttons to be read directly by the QSPI. The logic block shown in Figure 3 represents the glue logic necessary for the A/D control lines and the button status output.

PC Modes of Operation

There are two modes of operation for the console: real time control and programmed control. In real-time control, the user operates the system using the analog controls on the console to control the lights for a scene in real time. The console internally computes the output levels based on the levels of the faders on the input blocks. If a PC is connected, it will show the fader levels on the monitor.

In programmed control, the computer controls the lights based on pre-programmed cues that the user has defined. The computer polls the console for the fader data at regular intervals (about 50 times per second, just greater than the DMX output refresh rate) and stores this information in memory. It will then, depending on how the user has configured the software, do one of the following:

- Use the fader levels to determine the updated output
- Bypass the faders and output only the pre-programmed cues
- Enable both of these modes of operation at the same time, setting the output levels to whichever of the two modes' output levels would be higher

This last "combined" mode may require an example to illustrate. Supposedly the user has programmed a sequence that will flash several dimmers to full power, one at a time, and that this sequence will be used as an effect during a production. For testing purposes, however, the lighting designer may wish to test this sequence but at the same time keep the rest of the lights in the sequence on at 20% so where they are aimed can be seen. To accomplish this, one could run the programmed cue, while at the same time leaving the appropriate faders on at 20%. This way, when a dimmer would be at 100% as part of the sequence, it would still turn on 100%, but when the sequencer would turn the dimmer off, the fader would take precedence and the light would remain on at 20%.

Microcontroller Modes of Operation

The microcontroller will similarly have two modes of operation. In the normal PC-controlled mode, it will mainly serve the purpose of relaying data. The microcontroller will continuously sample the fader levels and pass this information through the USB link to the PC on command, and it will relay the output levels from the PC to the DMX512 output.

The second mode of control for the microcontroller will be a stand alone mode to accompany the PC's status monitor mode. In this mode of operation, the microcontroller will not only sample the fader data but also bypass the PC and output this data to the dimmers directly. This will allow the console to operate independently of the PC for troubleshooting purposes or for the convenience of the user.

Circuit Schematics

Page one of Appendix A contains the schematics for the USB peripheral chip, the RS-485 transmitter, and the input block address decoder. The USB chip was placed directly on the high half of the 16-bit microcontroller bus. It uses A1 to choose between the command and data ports on the USB chip. The main USB chip select is tied low since that is the only device existing at 0x800000, the memory location programmed into the read and write chip selects in the System Integration Module for this device. The 22Ω resistors are on the data lines for impedance matching. The LED is used for the "goodlink" feature of the Philips USB chip. It lights up when there is a valid USB connection and blinks when data is being transferred. The reset_n pin senses USB VBUS to detect the connection of a USB cable and the 1MegΩ resistor to ground is to leak this voltage when the connection is broken. The interrupt pin is open drain and needs a pullup resistor and is then connected directly to the IRQ pin in the microcontroller. The RS-485 transceiver is a MAX483 half-duplex transceiver. This takes the SCI output from the microcontroller and converts it to a differential signal for transmission. The input block address decoder consists of two 74154 4-to-16 decoders and a 74LS04 inverter. The decoder chips receive the 4 QSPI chip select lines from the microcontroller and an enable line. The enable line for the first 154 is the MODCLOCK pin from port E, and the second chip uses the inverse of this signal, effectively creating a 5-to-32 decoder. Each input block receives one output from each decoder for the faders and buttons.

Page two of Appendix A contains the schematic for the input blocks. Each block contains up to 8 fader controls (10kΩ slide potentiometers) and up to 8 momentary push buttons. The faders are connected to a MAX148 8-channel SPI-compatible serial A/D converter. The buttons are connected to a 74166 shift register. The serial output of the shift register is connected to a 74ALS373 latch, creating an SPI-compatible interface for the buttons. Both halves of the input block are connected directly to the SPI bus, which consists of a data input, a data output, and a serial clock. In addition, each input block receives one chip select for the faders and one for the buttons, as described above. The data output (MOSI) pin on the SPI bus is used to send commands to the A/D converter for fader sampling and to load the shift register for button sampling. Both sides of the block output on the data input (MISO) pin. When not selected, the input block goes into high impedance mode on the bus.

Microcontroller Software and Flowcharts

The microcontroller software is written completely in assembly language and is on the CD-ROM attached to this report. The flowcharts for the modules discussed here are in Appendix B. Assembly language was chosen over C in order to maximize speed and minimize size. The program is mainly interrupt driven, as described below. The object code is burned onto two EPROMs, which allows the system to start running at power up without having to load code from a host computer. A 3.5kB segment of internal RAM (TPURAM) is used as general-purpose memory for data storage. The two main data structures are a 512-byte output buffer for the DMX dimmer data and a 144-byte input buffer for the fader levels (128 bytes) and button states (16 bytes). The CPU is programmed to run at a system clock of 15.99 MHz, which, through clock division, provides a convenient way of generating the 250 kbps DMX clock.

The microcontroller software is broken up into several modules. They are organized into the following files:

usbdmx.asm	- main project file, contains global data and links all other files
vectable.inc	- hardcoded interrupt vectors for reset jump point and interrupt service routines
descriptor.inc	- hardcoded USB descriptor data structures used to identify and configure device
regs.inc	- equates for 68376 register names to their memory mapped locations
startup.asm	- initialization for microcontroller, memory, chip selects, SCI, QSPI, and USB
main.asm	- calculates output for 2-scene prototype board from fader inputs with cross-fade
sci.asm	- interrupt service routine for DMX data output stream
qspi.asm	- interrupt routine to read data from faders and buttons using QSPI
usb.asm	- interrupt service routine to handle all USB control and bulk transfer

The first 5 modules are fairly straightforward. The main routine takes the fader and button data from the input buffer and calculates dimmer levels for the output buffer. This routine operates the prototype board in 2-scene preset mode, which is a console setup with two rows of faders and a set of master faders. The top and bottom rows of channel faders control the X and Y scenes, and each scene has a master fader as well as a grand master fader (GM) for the entire system. The main loop reads the values for each channel in the X and Y scenes, combines them with the master levels, and calculates the resulting dimmer levels. The crossfade algorithm is as follows:

Step 1: Take the greater of the individual channels in the X and Y scenes

Step 2: Multiply the X scene channel level with the X master as a percentage and the Y scene channel level with the Y scene master. Add these two values.

Step 3: Take the lesser of step 1 and step 2. This is the final dimmer level.

The main loop algorithm also responds to the buttons underneath each channel fader. Pressing the top button turns that dimmer all the way on, and pressing the bottom button turns it all the way off. Currently, the "on" buttons take precedence over the "off" buttons.

The SCI interrupt service routine is given the highest priority of all the interrupt service routines so that the output can be refreshed as fast as possible. The routine begins by outputting a break. This part of the interrupt routine disables the SCI interrupt temporarily and enables the programmable interval timer (PIT) interrupt to go off after one PIT cycle, about 150us. On the next time through the interrupt routine, a mark consisting of at least two stop bits must be generated. Since the SCI does not have the capability to output two stop bits after a break, the routine turns off the transmitter, letting it idle high, but leaves the SCI interrupt enabled so that it will interrupt at the end of the frame that would have been sent out if the transmitter had been enabled. It then writes the start byte (0x00) and 512 data frames, each with one start bit and two stop bits. After the last data byte it starts the next break.

The QSPI interrupt service routine is reads the fader and button data into the input buffer. The QSPI module contains a command buffer to queue up to 16 serial transfers. Each A/D channel read requires two transfers, and with 8 faders in each block, one whole block is able to be read during each complete cycle through the command buffer. Each block also contains 8 buttons, which can all be read with one command buffer cycle. Thus there are 17 QSPI cycles required for one complete refresh of the input buffer. The module is programmed to operate at a serial clock rate of 333 kbps, which allows the input buffer to be refreshed slightly faster than the DMX output rate. A pin on port E (MODCLOCK) is used as a general output port to select between fader reading and button reading. The QSPI routine sets this pin low before reading faders and high before reading buttons.

The USB interrupt routine handles several things. The routine is entered whenever something on the USB peripheral chip changes. Two special cases are a bus reset and suspend change. A bus reset happens when the chip powers up or when the PC sends a device reset. A suspend change occurs whenever the chip has not received packets for 3ms or when the PC tells the device to enter suspend mode. The handler simply initializes a few variables on reset and ignores suspend changes. Since this is a self-powered device, tying the suspend pin on the chip to ground stops the device from entering suspend. If the data endpoints send or request data, the bulk endpoint routines are called. The out routine (out from the host, or PC, side) reads the data from the buffer and clears it, allowing it to accept more data. The in routine will place more data in the input buffer to be read the next time data is requested. The Philips USB chip only interrupts the in endpoint after data has been sent, so the in buffer must be primed during startup so that data is waiting when the PC requests it. Because of this, the data is always one packet of fader data behind real time. The control endpoint interrupts handle a whole other subset of functions. USB Specification Chapter 9 specifies several commands that a device should respond to. The most important of these are the set address command to set the device address when the system is configuring the device and the get descriptor commands to tell the host system how to configure the device. This device is a vendor specific device, so a few unnecessary commands, such as the set interface and set/clear feature, were not implemented. The device is still fully functional for its intended purpose.

Windows Software

There are two components of the PC software. The first is the driver and the second is the user application. The source code and binaries for the driver and the user application are also on the attached CD-ROM. The driver communicates directly with the USB stack in Windows while the user application then communicates with the driver. The driver handles initialization of the device in Windows and will create a handle to the device when requested to do so by a user application. For more information on Windows USB driver development, see the book Programming the Windows Driver Model or download the Microsoft Windows Driver Development Kit (DDK).

The user application simply mimics the fader panel in external control and writes the fader data directly to the DMX output buffer when in PC control mode.

Analysis of Results

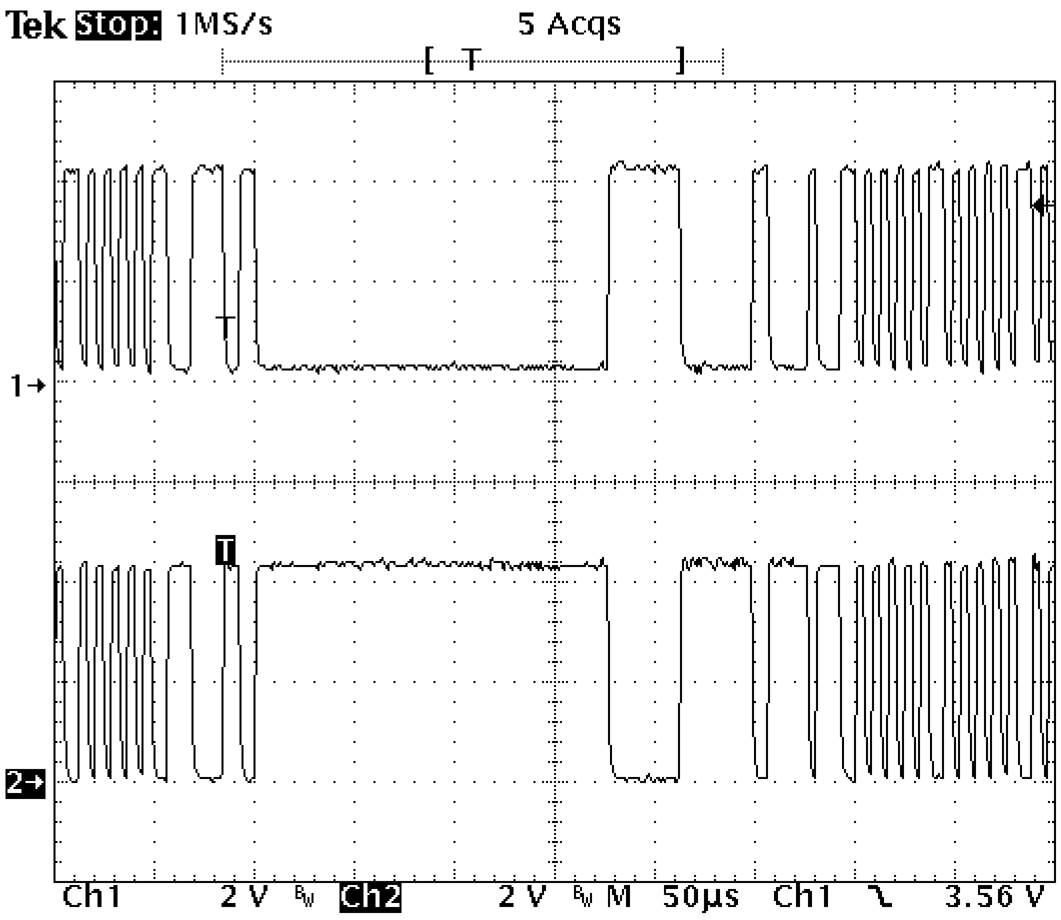

Figure 4 – RS-485 Differential Output

The RS-485 output is shown in the figure above. The top and bottom traces are the non-inverting and inverting outputs, respectively, using a +5V supply. The differential voltage ranges from -4V to 4V, well within the RS-485 specification of +/- 200mV. The MAX483 chip is slew-rate limited to 250kbps, which is the rate used in this application.

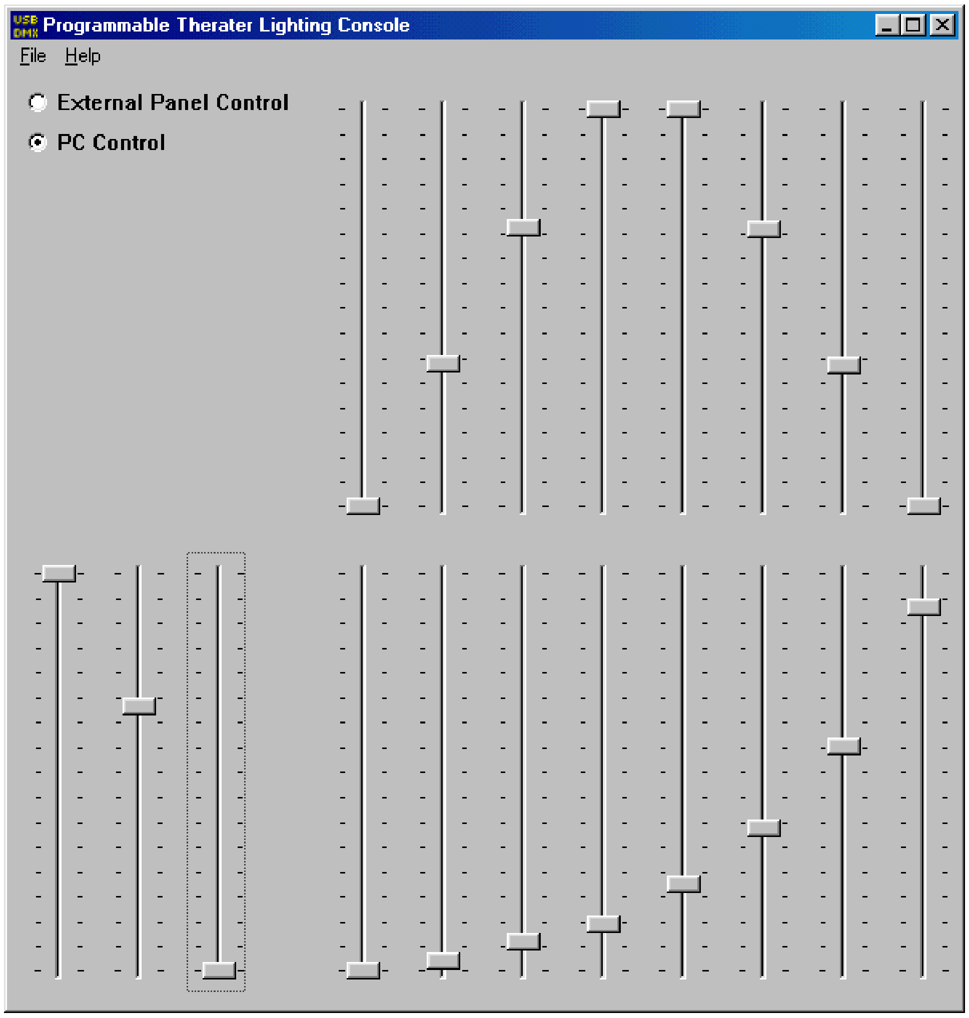

Figure 5 – User software in PC Control mode

In Figure 5, the faders were set using the external fader panel with the radio button in the upper left set to “External Panel Control”. The placement of the faders in this program matches that of the faders on the external panel. The software was then placed into “PC Control” mode as seen above so that Figure 6 could be captured. Currently, the user software outputs the master faders as normal dimmer channels, so they do not yet have the effect of actual master controls, as shown below.

Figure 6 shows the output of the system in PC Control mode with the levels set as shown in Figure 5. The three master faders can be seen output here as dimmers 1, 2, and 3. The output levels on the DMX stream are the inverse of what is shown on the screen due to the way the slider controls operate in Windows. This will be fixed when two-scene preset is implemented in the software.

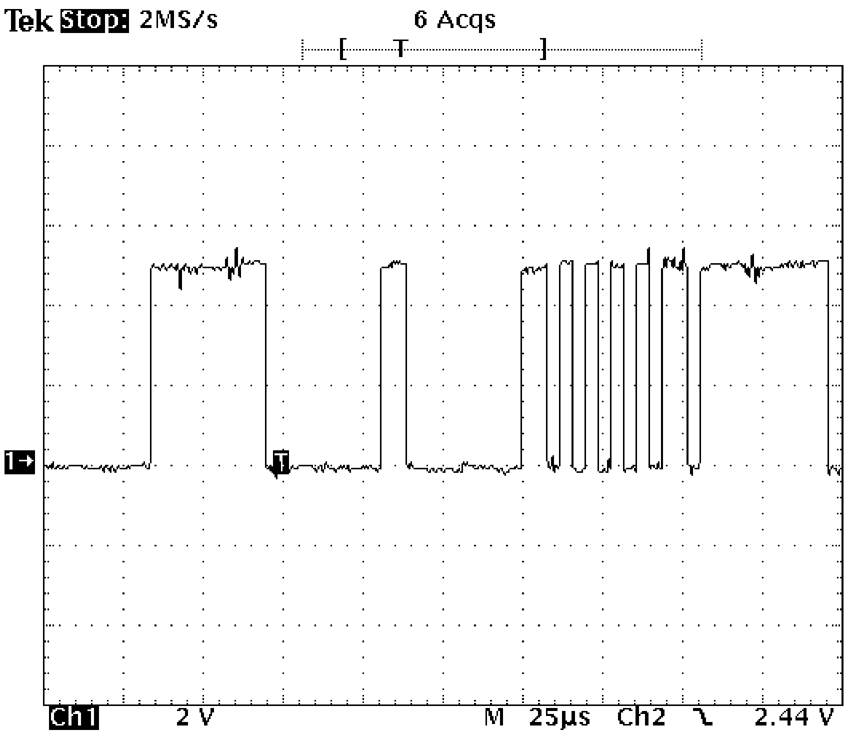

Figure 6 – Output of system in PC Control mode

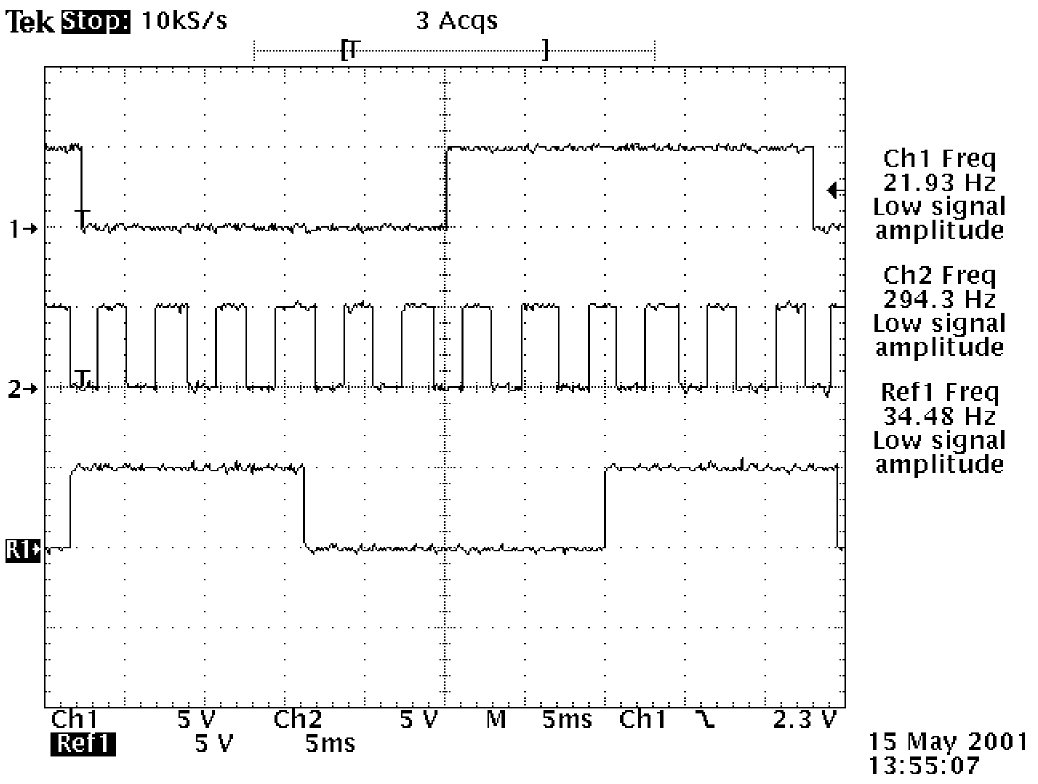

Figure 7 – Refresh rates of the system

Figure 7 shows the refresh rates of the DMX output stream (top), the input block sampling (bottom) and the main program loop (middle). These graphs were generated by flipping unused port pins during each of the three routines (SCI, QSPI, and main). Both the rising and falling edges represent one pass through the respective loops; the actual rates for each routine are double the frequency shown on the right side of the plot. Hence the DMX output is refreshing 43.86 times per second, which is very close to the theoretical maximum of 44.11 Hz.

Conclusions

The project goal was achieved. The hardware design and firmware are complete. The USB hardware and software (both PC and console side) are fully operational and debugged. There are a few USB routines on the console that are specific to this device, such as the bulk read and write routines. Generalizing this, as well as implementing the few unused and unimplemented USB control functions could be done as a future extension to this project. The user software is functional, although at the current time there is only a minimal feature set. Adding capabilities to this user software, such as programmed cues and capturing data to be played back at a later time, is another area for expansion to this project. The Bradley Electrical Engineering Department will be able to use the USB module (consisting of the hardware, console firmware, windows driver, and windows user application) as a framework for other projects in coming years.

References

Motorola 68376 User Guide, Motorola

Motorola CPU32 Reference Manual, Motorola

DMX512 Digital Data Transmission Standard, 1990 Revision. United States Institute for Theatre Technology

Universal Serial Bus Specification Revision 1.1, USB Implementers Forum

Walter Oney, Programming the Windows Driver Model, Microsoft Press

Microsoft Windows Driver Development Kit, http://www.microsoft.com/ddk/

Microsoft MSDN Library Online, http://msdn.microsoft.com/library/default.asp

Appendix A

Appendix B