Real-time Lighting System for Large Group Interaction

Joshua Randall

Massachusetts Institute of Technology
May 2002

Abstract

Lighting systems have historically been controlled by an individual or small group of human operators working together in real-time. Applications for real-time lighting control include theatre, concerts, dance performances, and dance parties such as raves. For the first three applications, changes in the lighting are usually designed to be triggered by premeditated actions of the performers on-stage. In those cases, control by a small number of operators is possible since they need only to pay attention to the relatively small number of performers in order to achieve the goals of the lighting design. In an environment such as a rave, the primary focus is a large number of people dancing together in an unpredictable way. In such a dynamic environment, control by human operators is not ideal, since it is difficult for a small group to follow changes in the movements of a large group, or to react quickly enough to sudden changes in group activity. This paper will present a system which makes it possible for large groups, such as dancers, to interact with the lighting system directly.

1 Introduction

The real-time lighting control system creates an interaction between a large group of dancers and the dance lighting. It depends upon another, already existing, system to provide the input data. The "system for large group musical interaction using disposable wireless motion sensors," described by Feldmeier et al.[1], collects motion data from the dancers in real-time and derives parameters used for synthesizing electronic music. These parameters are also useful for lighting control, and are supplied digitally to the lighting system. Refer to Figure 1 for an overview of the interconnection between systems and subsystems.

The lighting controller takes these parameters as input and generates a set of lighting parameters as output. Programmable mappings between the input and output parameters allow the designer to determine how group activity translates into lighting effects. These effects are achieved by making changes to the lighting control data which are sent to the lighting output interface.

The lighting output interface is attached to the lighting controller and to the lighting equipment. Data received from the controller is transmitted to the lighting devices using an industry-accepted standard protocol.

The lighting system consists of both hardware and software elements. Hardware is needed to interface with lighting equipment, while software drives the hardware and performs the input to output mappings. This paper presents the technical design of the hardware and software.

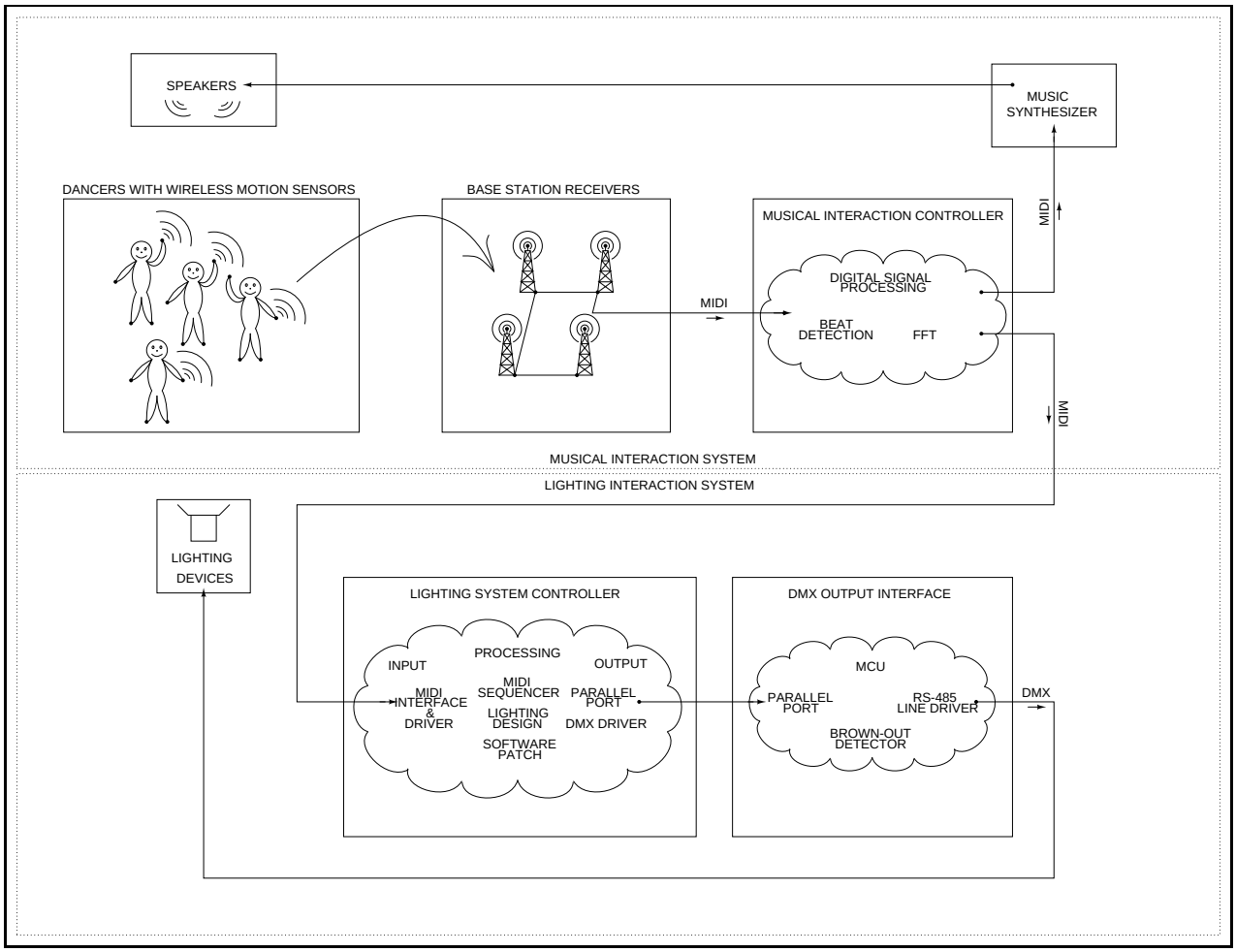

Figure 1: Large Group Interaction System Overview.

1.1 Hardware

An IBM-compatible personal computer (PC) is the core piece of hardware in the lighting system. It receives input indirectly from the wireless motion sensors by way of the large group musical interaction system. This system sends data to the PC over the Musical Instrument Digital Interface (MIDI) port. These data received over the MIDI port are mapped to lighting data on the PC, and the output is via a DMX512 interface connected to the parallel printer port. The DMX512 digital data transmission standard (DMX protocol) is the most commonly used method for communicating between lighting control systems and theatrical lighting devices such as dimmers, color scrollers, and moving lights. The standard

is supported by the United States Institute for Theatre Technology and has been adopted by most of the theater lighting manufacturers in the United States and throughout the world. While many DMX512 computer interfaces are commercially available, they are unnecessarily costly. The design of an inexpensive DMX512 compliant interface for the PC parallel port was therefore a major portion of this project and will be presented in this paper.

1.2 Software

The software for the lighting system consists of several components, which can be divided into three modules: the input protocol handler, the controller which processes the data and performs the mapping between input and output, and the output driver. Because access to the input and output ports is not handled uniformly across operating systems, the input and output components had to be written for a specific operating system. The operating system chosen is GNU/Linux because it is reliable, freely available, and has free drivers for both the input and output ports. All software is written in the C programming language, but each module is separate and any module could be rewritten in another language if so desired. The controller module is the domain of the lighting designer, and is the only module that should need to be changed by the designer. While C is perfectly valid choice for programming a design, other programming environments such as C++ or LISP might be more desirable for more complicated designs. The realization of a design in this system consists of developing a control module that contains all the different lighting "looks" and defines how changes in them should occur depending on input parameters.

2 Musical Interaction System

Wireless motion sensors are attached to dancers and transmit acceleration data in real-time. The musical interaction system [1] uses base station receivers to collect data from the many wireless sensors attached to dancers (see Figure 1). A computer processes the data from the sensors, using algorithms such as beat detection and fast-fourier transforms (FFTs). Derived parameters such as intensity and rhythm data are sent on the MIDI output line. MIDI is a set of communications protocols used by musical instruments to exchange musical and timing data, which makes it ideally suited for this purpose. The MIDI standard defines both the communication protocols and the physical hardware parameters for cabling and interfaces. The MIDI output line is connected to a music synthesizer which uses the derived parameters to generate music that is played over speakers on the dance floor. The dancers' movement in reaction to the music creates a feedback loop that can either lead or follow the dancers' actions [1]. The MIDI output line from the musical interaction system is also connected to the MIDI input port of the lighting controller so that the derived parameters can be used to provide a lighting feedback loop.

2.1 Hardware

2.1.1 Wireless motion sensors

The wireless motion sensors consist of small, inexpensive, wireless transmitters that send short bursts of RF energy whenever they sense acceleration above a threshold level [1]. The transmitter is triggered by a piezoelectric film cantilever. The strength of the RF burst is kept to a minimum to conserve energy and so that each transmitter has a 3-10 meter transmission radius. As a result, only nearby receivers register the burst. This is desirable since it imparts some information about the location of the sensor.

2.1.2 Base station receivers

Several base station receivers are physically distributed across the dancing area. For a small dance area, an array of four receivers located at the corners could be used. Each receiver counts the number of bursts received from the wireless motion sensors within a short window of time. Since no additional information is encoded in the RF burst, the number of bursts received per unit time is the only data collected by the receivers. This data is periodically transmitted over the MIDI line to the musical interaction controller.

2.1.3 Musical Interaction Controller

A Macintosh computer with MIDI input and output ports is used by the large group musical interaction system. It performs data processing on the acceleration data collected from the receivers and outputs derived parameters on the MIDI output port.

2.1.4 Synthesizer

An electronic music synthesizer receives derived parameters from the musical interaction controller and generates music which is played on the dance floor.

2.2 Software

2.2.1 Motion data analysis

The large group musical interaction system uses the MAX programming environment to perform data processing algorithms on the MIDI input stream from the base station receivers (see Figure 2). MAX is a Macintosh based, object-oriented programming environment that has built in MIDI support and is well suited to analyzing data in real time. FFTs are used to work with the motion data in the frequency domain. Many parameters are derived, such as intensity, frequency, and rhythm (beat detection). The derived data is transmitted over a MIDI output stream which goes both to the lighting system and to the electronic music synthesizer.

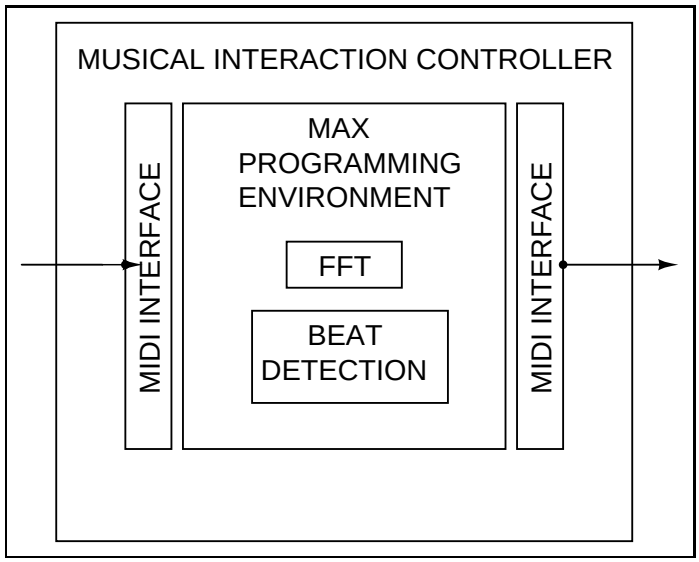

Figure 2: Musical interaction controller block diagram.

3 Lighting Controller

The controller consists of software on the PC that performs the mapping between derived parameters received from the large group musical interaction system and lighting parameters, which are represented by channel levels. The code for the mapping is written by the lighting designer for a specific event. This is necessary since the functions of the lighting channels will vary depending on the lighting devices used, and because designers will want the controller to behave differently depending on the style of dance, music, and physical layout of the dance space.

3.1 Hardware

An IBM compatible personal computer (PC) is the core of the system and is used primarily to map inputs to outputs. The hardware used as the test platform has an Intel 486 processor and 32 MB of system memory (RAM). The lighting system receives MIDI data from the large group musical interaction system via the MIDI interface (MPU401 port) built-in to the sound card on the PC.

3.2 Software

The core module receives data from the input protocol handler (MIDI driver) and transmits data to the output module. The state of the output is maintained in memory as an array of channel levels. This array is modified based on the input received and the mappings which are programmed into the core module. The useful input data are the beats and the activity level of each of the areas. Beats from each area synchronize lighting changes in that area. The intensity level determines how much change occurs with each beat. For example, a low intensity level in one area may indicate that no dancing is occurring there, so lighting changes

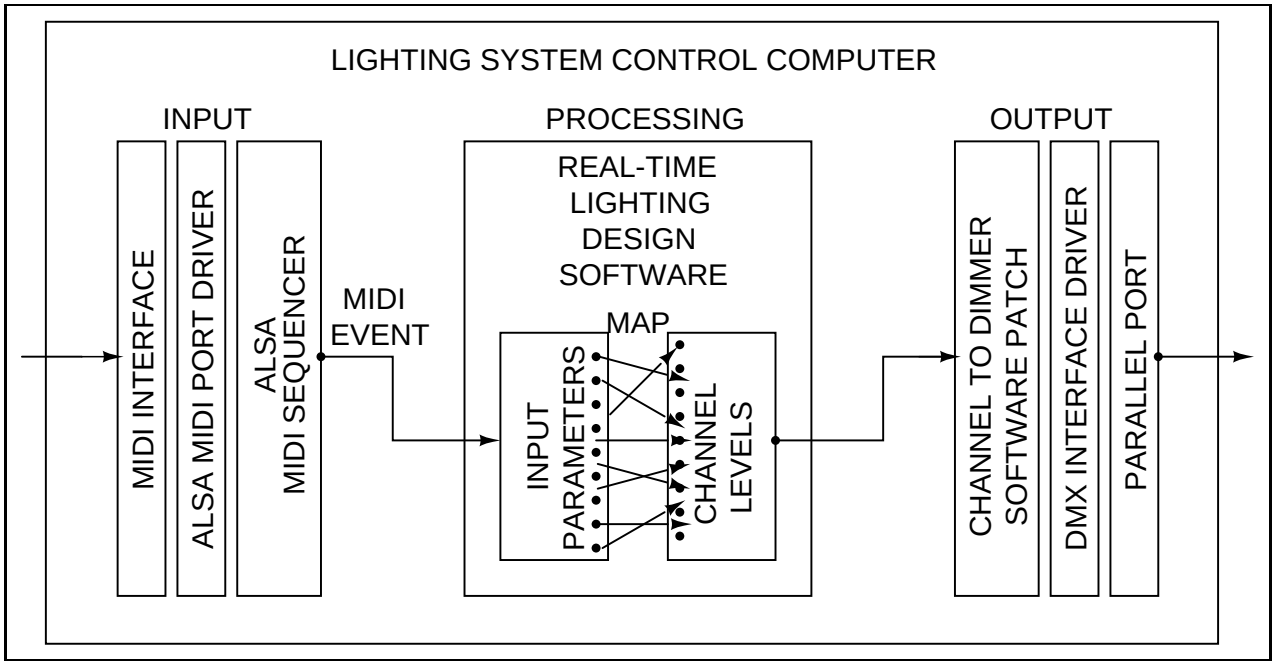

Figure 3: Lighting controller block diagram.

in that area only happen once every few beats. At the opposite end of the spectrum, many dancers moving at the same time in an area could cause several different lighting changes in a single beat. Setting up the exact relationship between the movement intensity, the beats, and the light parameters (such as intensity, color, and position) is up to the designer of the specific event.

3.2.1 MIDI driver

A number of drivers are available for the Linux kernel that provide access to the MIDI port used as the input component to the lighting system. The Advanced Linux Sound Architecture (ALSA) provides sound card and MIDI port drivers as well as a MIDI sequencer and a library for accessing it. The input module uses the ALSA sequencer client API to receive MIDI events in real time as they arrive. Data is then extracted from the MIDI event and passed to the core module for mapping.

4 Lighting Output Interface

The lighting system transmits output via a DMX512 output interface for the PC. The DMX protocol is an asynchronous serial protocol that runs at a data rate of 250 kilobits per second (kbps) over a balanced RS-485 transmission line. The protocol consists of a reset signal followed by a data packet. A series of 8-bit values representing each dimmer level is transmitted sequentially in the data packet beginning with dimmer number 1 and incrementing to the highest dimmer number used (up to a maximum of 512) [2].

DMX-capable lighting devices interpret the levels of the dimmer number(s) to which they are set to respond. The dimmer number used by each device is usually set by the end user as the DMX "address" of the device. Each address controls a parameter of a lighting device. Some lighting devices have only one address, while others use twenty or more addresses to control their many parameters. For example, an actual lighting dimmer uses the level of a single address to set the intensity of the lighting instruments connected to it, while an advanced moving light responds to many different addresses, using their levels to set parameters such as pan, tilt, intensity, color, pattern, and rotation speed.

Since commercially available DMX512 interfaces are very costly, ranging from $250 to $500 or more, an inexpensive interface has been designed and built for this project. The interface consists of hardware (see Figure 4) that is dedicated to transmitting dimmer levels to the DMX port and software drivers for the PC that provide functions for sending dimmer levels to the hardware.

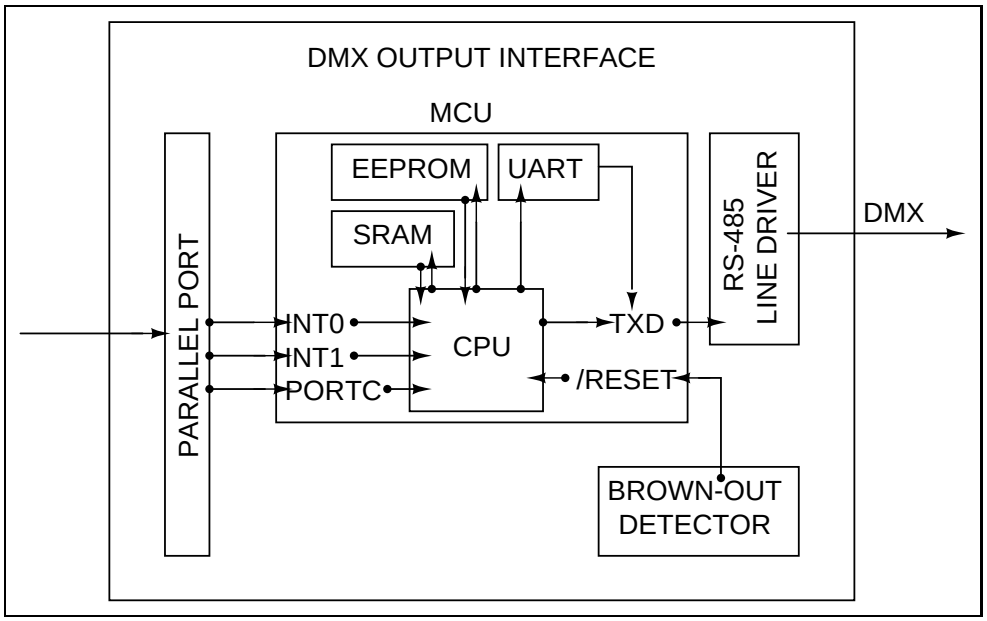

Figure 4: Lighting output interface block diagram.

4.1 Hardware

The interface is designed to fully comply with the USITT DMX512/1990 standard while being inexpensive and relatively easy to build. It would be possible to implement the hardware using a finite state machine programmed into a field programmable gate array (FPGA) or other type of programmable logic device. However, this would require a rather large number of gates and separate chips for memory, input, and output which would add considerable expense and unnecessary complexity to the design. A much simpler and less expensive design is based on a microprocessor control unit (MCU) (see Table 1) [3].

MCUs are versatile, relatively low cost, and can be easily upgraded with new programming in the future. They are often packaged with various memory and input/output (I/O)

Table 1: DMX interface circuit component cost [3].

Component	Single-Unit Cost	Minimum Cost at Quantity (100+)
AT90S8515-8	$8.51	$5.25
MAX485 CPA	$2.76	$1.33
DS1813-5	$1.11	$0.58
8MHz Crystal	$1.90	$1.90
5x2 Header Connector	$1.17	$0.41
13x2 Header Connector	$4.28	$1.67
3x1 Header Connector	$2.24	$0.79
TOTAL	$21.97	$11.93

units on-board. The MCU chosen for the DMX output interface is the Atmel AV90S8515 8-bit RISC microcontroller [4]. It runs at a maximum of 8MHz; has 8 kilobytes of In-System Programmable Flash program memory, 512 bytes EEPROM, 512 bytes SRAM, a programmable serial UART, and two external interrupts for a cost of under $9 (see Table 1 for the cost of the major hardware components). Because of its many on-board features, only two other semiconductor devices are necessary to complete the hardware for the DMX interface: an RS-485 line driver and a brown-out detector. The line driver is necessary to meet the electrical characteristics required by the DMX standard, and the brown-out detector is used to protect the microprocessor from a low-voltage power supply such as would likely be present during power-up. Please refer to Figure 5 for a complete circuit diagram and Table 2 for a description of the purpose of the major components.

Table 2: Purpose of DMX interface major circuit components.

Circuit Label	Component	Purpose
U1	AT90S8515-8	Microprocessor Control Unit
U2	MAX485 CPA	RS-485 Line Driver
U3	DS1813-5	Brown-out Detector
X1	8MHz Crystal	MCU Clock
J1	5x2 Header Connector	In-System Programming Port
J2	13x2 Header Connector	Parallel Port
J3	3x1 Header Connector	DMX Output Port

On power-up, the MCU loads the contents of the nonvolatile EEPROM, which is used to store the default dimmer levels, into the SRAM. The SRAM is used to store the current dimmer levels and is updated via the parallel interface which is implemented using the MCU's general-purpose I/O ports and external interrupts (see U1 pins PA0, PA4, PC0-7, PD4-5, and INT0-1 in Figure 5). The serial UART is set to 250 kbps by using an 8MHz system clock divided by 32. The UART is used to transmit the DMX protocol data packets, while the DMX reset timing is sent synchronously by the microprocessor. Both the UART and the microprocessor output DMX data on the TXD pin which is connected to the input of the

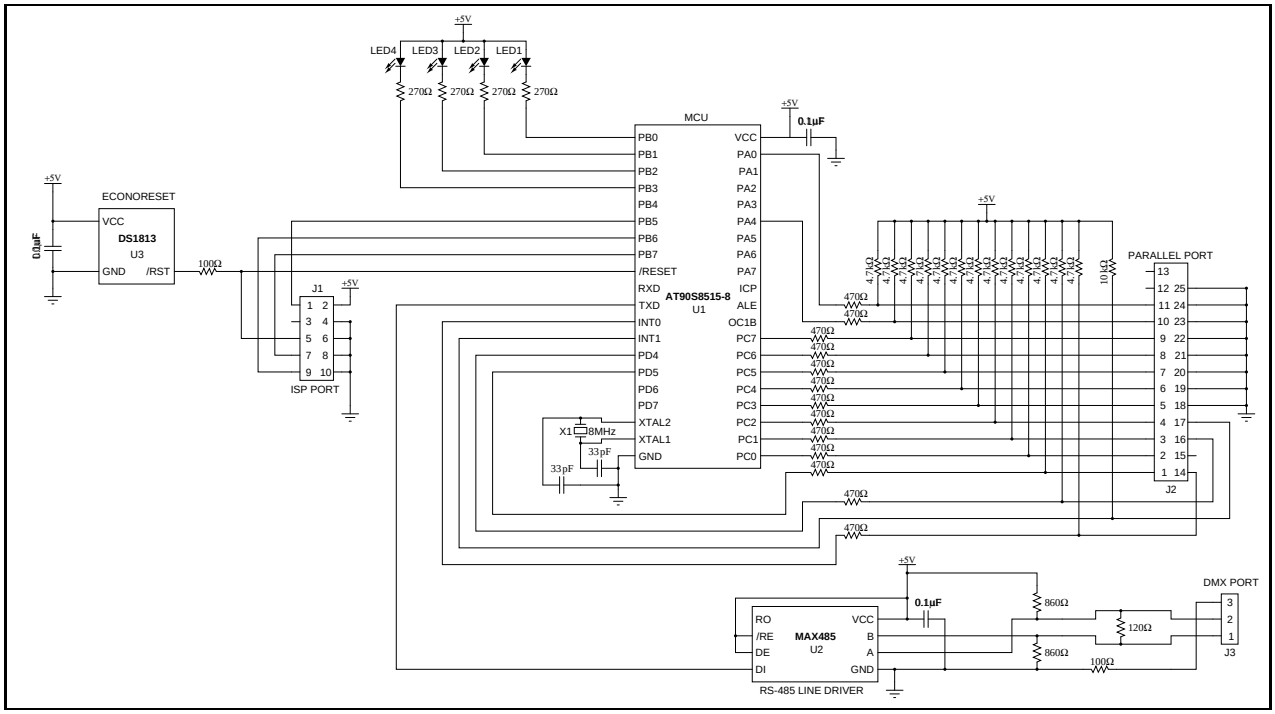

Figure 5: DMX interface circuit diagram.

RS-485 line driver. An in-system programming port allows the MCU to be reprogrammed with updated firmware should additional features be desired or if bugs need to be fixed (see J1 in Figure 5). Finally, a set of four LEDs connected to I/O port B (PB0-PB3) displays the current status of the device. See Table 3 for a description of LED purpose.

Table 3: Purpose of LEDs on the DMX interface.

Name	**Purpose**
LED1	Indicates DMX output idle.
LED2	Indicates DMX output active.
LED3	Indicates EEPROM reading or writing.
LED4	Indicates parallel port activity.

The MAX485 chip receives the serial bitstream from the TXD pin on the MCU and drives the RS-485 differential transmission line, with the help of external pull-up, pull-down, and line loading resistors. Refer to Figure 6 for resistor values in the line driver circuit.

The DS1813 5V EconoReset acts as a brown-out detector. It monitors the power supply using a temperature reference and comparator to detect out-of-tolerance conditions [5]. The output of the EconoReset is connected to the MCU's /RESET pin so that the microprocessor will be disabled if a brown-out occurs on the power supply.

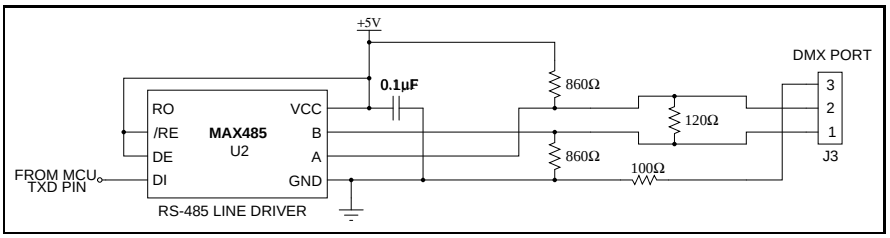

Figure 6: Line driver circuit diagram.

4.2 Software

Software for the DMX output interface consists of firmware on the MCU and a driver for the PC.

4.2.1 MCU Firmware

The firmware for the MCU is written in C and compiled to AVR microcode. In its normal state, the firmware transmits DMX packets, sending around 40 complete updates per second. The dimmer levels are read out of an array stored in the SRAM (see Figure 7) one at a time and transmitted via the UART. The DMX reset timing is generated by the dmx_reset function seen in Figure 8.

```
/*****************************************************************************************************************
/* Transmit DMX Packet
/*****************************************************************************************************************/
void dmx_send() {
  LED2ON;               /* LED2 on during DMX activity */
  LED1OFF;              /* LED1 off during DMX activity */
  if(DMX_CURDIMMER_TX == 0) /* Send Reset Sequence */
    {
      delay(240);               /* Additional Sleepiness during MBB (~200us) */
      PORTA |= DMXBUSY;   /* Set DMXBUSY */
      dmx_reset();              /* Transmit Reset */
      TransmitByte(STARTCODE);  /* Send START code */
      DMX_CURDIMMER_TX++;  /* Increment transmit counter */
    } /* end if(DMX_CURDIMMER_TX == 0) */
  else /* Data slot 1->DIMMERMAX */
    {
      TransmitByte(DMX_LEVELS[DMX_CURDIMMER_TX-1]); /* Send byte to RS485 Line Driver */
      if(DMX_CURDIMMER_TX < DIMMERMAX) /* Data slot 1->(DIMMERMAX-1) */
        {
          DMX_CURDIMMER_TX++;   /* Increment transmit counter */
        } /* end if(DMX_CURDIMMER_TX < DIMMERMAX) */
      else if(DMX_CURDIMMER_TX == DIMMERMAX) /* Last slot in packet! */
        {
          DMX_CURDIMMER_TX = 0;  /* Reset transmit counter */
          LED1ON;                /* LED1 on during DMX idle */
          LED2OFF;               /* LED2 off during DMX idle */
          PORTA &=~DMXBUSY;      /* Unset DMXBUSY */
        } /* end else  if(DMX_CURDIMMER_TX == DIMMERMAX) */
    } /* end else Data slot 1->DIMMERMAX */
} /* end dmx_send() */
```

Figure 7: Firmware source code for DMX data packet transmission via UART.

An edge detected on the INT0 pin (DATASTROBE signal from the PC) triggers an interrupt handler that receives data from the parallel port and stores received dimmer levels in the SRAM.

```
/*****************************************************************************************************************
/* Transmit DMX Reset
/*****************************************************************************************************************/
void dmx_reset() {
  PORTD |=TXDPIN;           /* Write port D bit 1 (TxD) to HIGH */
  UART_TRANSMIT_OFF();    /* Disable UART transmit (so we can transmit break timing directly) */
  /* "MARK" Before BREAK (MBB) = min 0s, typical --, max 1s */
  /* No delay needed */
  PORTD &=~TXDPIN;          /* Write port D bit 1 (TxD) to LOW for BREAK */
  /* "SPACE" for BREAK = min 88us, typical --, max -- */
  /* Wait ~90us */
  delay(120);
  PORTD |=TXDPIN;           /* Write port D bit 1 (TxD) to HIGH */
  /* "MARK" after BREAK (MAB) = min 8us, typical --, max 1s */
  /* Wait 9us = 24*3 = 72 cycles for MAB */
  delay(10);
  UART_TRANSMIT_ON();      /* Enable UART Transmit */
} /* emd dmx_reset() */
```

Figure 8: Firmware source code for synchronous DMX reset timing.

4.2.2 Driver

The protocol between the DMX512 interface and the PC has been designed to allow continuous transmission on the DMX line. When DMX data is being transmitted, the interface asserts the DMXBUSY signal so that the driver will not interrupt a DMX transmission. When the driver wants to send data to the interface, it waits until the DMXBUSY signal goes low before it sends the DATASTROBE to transfer a byte on the DATA pins. It then waits for the DMXBUSY signal to go high again before attempting to send the next byte (see Figure 9). The driver provides a C function for sending an array of dimmer levels. dmxsend(char dimmer_level[]) sends levels in the dimmer_level[] array to the interface.

5 Conclusion

The real-time lighting system for large group interaction makes it possible for dancers to interact with the lighting system. Movements by dancers trigger changes in the lighting of the dance area that are determined ahead of time by the lighting designer. While the lighting as a whole will react to group activity, participants in the dance should be able to tell that their motions are triggering lighting changes, since even low intensity signals can be designed to make subtle changes, perhaps changing a single lighting instrument.

The software designed for this project allows for flexibility by making the controller a separate module that is programmed by the lighting designer for a specific event. The complexity of the mapping is therefore virtually unlimited, and the design is constrained only by the maximum number of dimmers controllable by the interface (which is set forth in the protocol specification).

The hardware interface designed and built for this project allows a PC to transmit lighting control data at high speed. The major circuit components cost more than 10 times less than commercially available interfaces. Even if the other components such as the circuit board, case, and power supply were to double the cost of the interface, it would still amount to only 20% of the cost of a commercial unit.

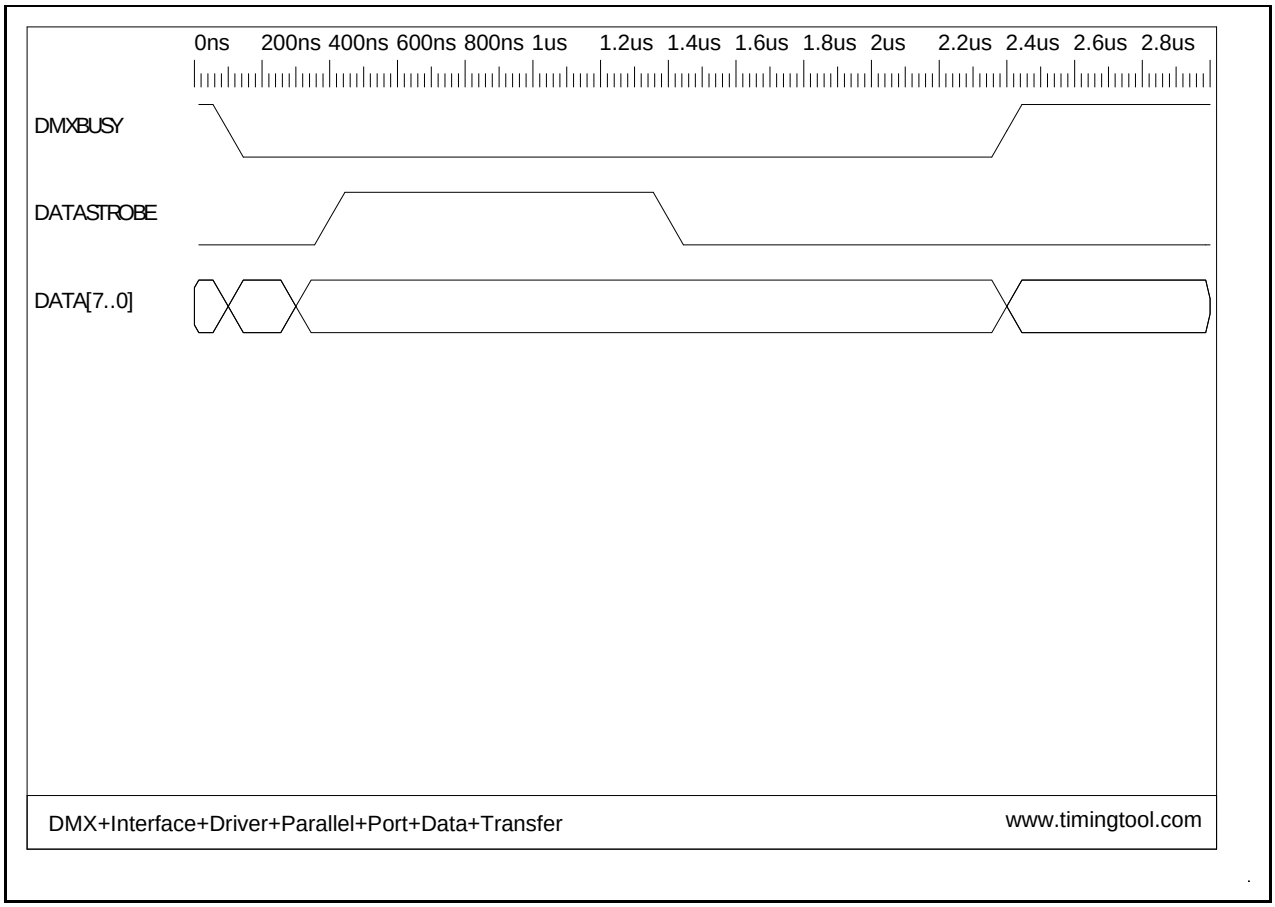

Figure 9: DMX interface parallel port data transfer protocol.

References

[1] Feldmeier, Mark; Paradiso, Joseph A.; Malinowski, Mateusz. (2002). *Large Group Musical Interaction using Disposable Wireless motion sensors.* Responsive Environments Group, M.I.T. Media Laboratory. Cambridge, MA.

[2] USITT Engineering Commission. (1990). *DMX512/1990 Digital Data Transmission Standard for Dimmers and Controllers.* United States Institute for Theatre Technology, Inc. New York, NY.

[3] Digi-Key Corporation. (2002). *Digi-Key May-August 2002 Catalog.* http://info.digikey.com/T022/V5Catalog.html

[4] Atmel Corportation. (2001). *8-bit AVR Microcontroller with 8K bytes In-System Programmable Flash.* http://www.atmel.com/atmel/acrobat/doc0841.pdf

[5] Maxim Integrated Products. (2002). *5V EconoReset with Pushbutton.* http://pdfserv.maxim-ic.com/arpdf/DS1813.pdf