SHEET MUSIC GENERATOR: MIDI-PC INTERFACE

A Design Project Report

Presented to the Engineering Division of the Graduate School

Of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

By

Meg Walraed-Sullivan

Project Advisor: Dr. Bruce Land

Degree Date: August, 2004

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Sheet Music Generator: MIDI-PC Interface

Author: Meg Walraed-Sullivan

Abstract:

This project entails a fully functional interface between a standard Musical Instrument Digital Interface (MIDI) device and a personal computer, including a software application for processing and display of MIDI data. The hardware interface uses an Atmel Mega32 microcontroller to facilitate communications between a MIDI device and computer. The microcontroller receives MIDI data through a standard MIDI cable, filters and encodes the data, then sends packets to the PC via a serial UART connection. Educational aspects of this project include the mastery of a programming language and corresponding package for graphics rendering, the design of a functional and intuitive software application, the creation of appropriate hardware to implement the interface between two devices, and the design of efficient firmware to manage all hardware.

Report Approved by

Project Advisor: _____Date:_____

Executive Summary

The goal of this project is to construct a fully functional interface between a standard MIDI device and a personal computer operating with Windows XP. The solution to such a problem can be divided into several sub-goals, including the design of a hardware circuit to interface with both the MIDI device and PC, the creation of appropriate firmware to process, filter, and encode MIDI output and forward information to the PC, and the design and generation of a software application to receive, manage, process, and display MIDI data in a convenient and intuitive manner. The hardware consists of an Atmel Mega32 microcontroller for manipulation and forwarding of MIDI data, a simple circuit to isolate the MIDI connection from other hardware, and an RS232 interface to the personal computer. The MIDI processing firmware for the Mega32 microcontroller is written in AVR Code Vision C, and encodes information received from the MIDI device into packets to be sent to the PC. The software application for the display and user manipulation of MIDI data is written in C++ using Microsoft Foundation Classes for graphical output and receives data from the hardware through a serial UART interface.

Design Problemand Requirements

The tedium of creating a software representation (sheet music) of a musical composition remains a large problem for musicians today. Two primary alternatives for this task exist. A musician can choose to enter notation manually, using a computer keyboard and mouse. However, this choice requires that the user purchase the appropriate software for this task, and the use of such software is often tiresome and time-consuming. This option also removes part of the musical aspect of the creative process of music composition by eliminating the possibility of using a musical instrument for composition. A preferred alternative would allow for the entry of musical notation via a musical instrument. Since the composition is to be stored and manipulated on a computer, a digital musical device is a natural choice for a musical instrument to use.

Solutions which facilitate the entry of musical notation via standard Musical Instrument Digital Interface (MIDI) devices do exist, but they prove to be as tedious, expensive, and difficult to use as do solutions involving solely the use of a computer. The difficulty inherent in using such a tool stems primarily from the expense and time required in order to begin using such a product. A user must purchase the appropriate software, sometimes at costs which range in the hundreds of dollars, and must own a compatible MIDI device and have a compatible sound card installed in his or her personal computer. Not surprisingly, this feat is not easily accomplished. It is often quite costly and extremely time-consuming to set up such a configuration, if one can be found. However, many musicians endure these trials, as alternative methods are few

and far between. It is therefore desirable to create a package which accomplishes this task without enforcing such expensive and unattainable requirements. The solution created must include adequate software for the manipulation and display of the musical notation, must be able to interface with any standard MIDI device, and must also interface seamlessly with most personal computers.

It is clear that the use of a microcontroller would be applicable to this problem; such a solution would delegate management of the interface between the PC and MIDI device to the microcontroller and surrounding hardware, thus greatly reducing compatibly requirements. This solution would also allow for the creation of more generic software for data display and manipulation, thus decreasing the cost of the system as a whole.

This project sets out to address the task of employing an inexpensive microcontroller in an interface between a personal computer and standard MIDI device. The tasks relevant to this process include:

- The design and implementation of a software application for graphic rendering, to be used to display a musical composition as it evolves.
- The design and implementation of a software application capable of receiving and processing serial data originally sent from a MIDI device.
- The design and construction of hardware capable of physically connecting the MIDI device to the personal computer.

- The design and implementation of firmware to control and manage the interface hardware.

The following discussion documents the design processes of all aspects of the goals mentioned above.

Design and Implementation

Details of the design and creation of this system can be separated into three distinct sub-categories: hardware, firmware, and software. Each of these is discussed in turn below.

Hardware

The high-level design of the hardware used in this project consists of a multi-step connection between a standard MIDI keyboard and a serial UART-16550 port on a personal computer. MIDI packets travel from a standard MIDI device to a microcontroller where they are received, processed, and filtered. The newly encoded packets then travel through a simple RS232 serial interface and standard serial cable to a COM port on a personal computer. A software application receives this serial data and processes it, displaying results onscreen to the user.

The hardware used for this project was originally constructed using an AVR SDK 500 development board. The development board provides conveniences such as push buttons, LED's, and an RS232 interface, with only the connection of a set of jumpers. As the project progressed, it was moved to a breadboard and components previously supplied by the development board were replaced with those discussed below. A straightforward following step would be to construct a printed circuit board for the entirety of the hardware, thus allowing for mass production. However, the expense and time required for completion of this step forces the creation of a printed circuit board outside of the scope of this project.

The crucial element of this hardware configuration is the Atmel Mega32 microcontroller, shown adjacent. This processor runs at 16MHz and offers four Input/Output ports and an on-chip UART, among many other convenient and valuable features. The processor runs at 5 volts, powered by a 9 volt battery (or other DC power source) connected through a voltage regulator. The battery is connected to the voltage regulator through a switch so as to minimize power consumption when the device is not in use. The Mega32 is driven by a 16MHz external oscillator; however, it should be noted that it is also feasible to use a simple crystal to drive the chip. This solution was not employed as the capacitance of the breadboard used can interfere with proper oscillation of a crystal. Packets from the MIDI keyboard are received via the on-chip UART of the Mega32 and are then forwarded on with a software UART, detailed below in the section entitled "Firmware."

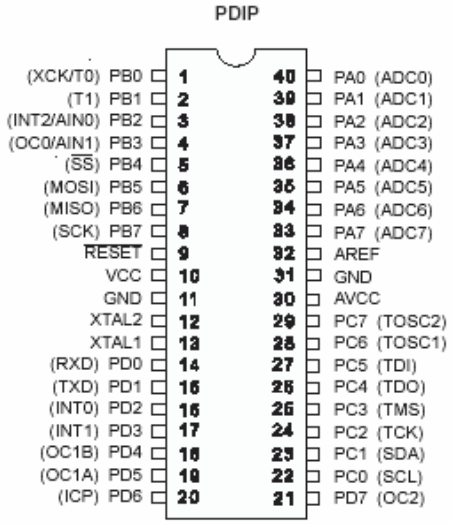


Figure 1: Mega32 Pinout

The MIDI device is connected to the microcontroller through an optoisolator, so as to minimize the effects of ground loops on the circuit. The output of this optoisolator runs to pin 0 of PORT D, one of the microcontroller's four Input/Output ports, where it is handled by the on-chip UART.

Upon passing through the microcontroller, encoded MIDI data must be forwarded on to the PC for processing and display. This is accomplished through a serial connection using a software UART written for the microcontroller and a standard COM port on the PC. The software UART is implemented in firmware and communicates through an external interrupt (PORT D, pin 2) and a generic port pin (PORT D, pin 3). A Max233 voltage level shifter converts values generated by the UART into those appropriate for serial UART communication and sends the resulting data through a 9 pin female D-Sub connector. A full schematic of the hardware configuration is shown below.

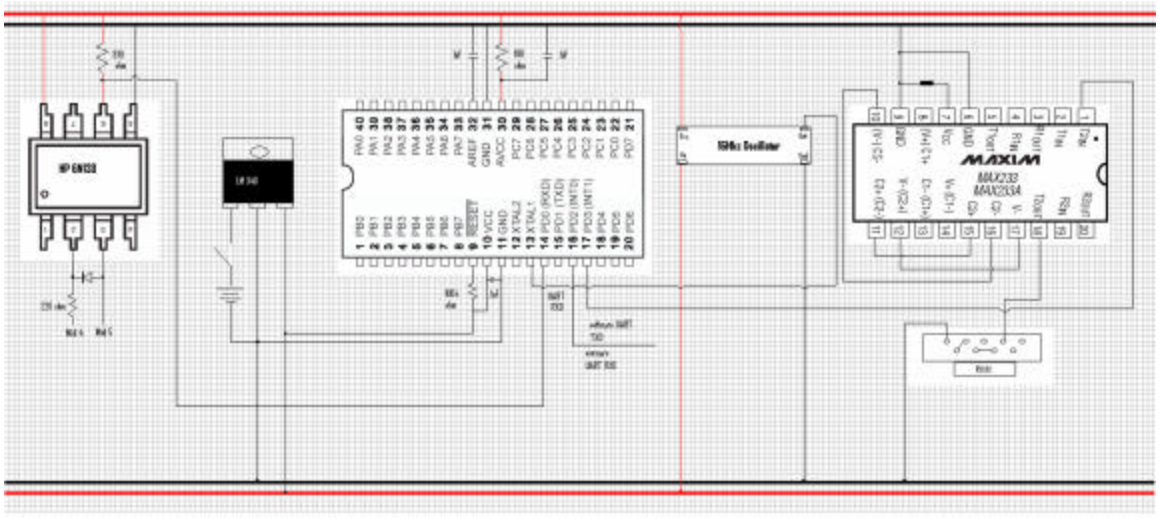


Figure 2: Hardware Schematic

The microcontroller proves to be an inexpensive and convenient solution to the problem of interfacing two hardware devices. The cost of the chip and surrounding hardware is relatively low and would be less still if mass production were to be explored. Another cost under consideration is that of labor. Since this particular microcontroller is

generally straightforward and intuitive to interface with, it remains true that the Mega32 allows for the creation of an inexpensive but efficient solution. The following listing of all hardware used details the cost of this portion of the project.

Part Description	Part Number	Cost/Item	Quantity	Total Cost
Optoisolator	HP 6N138	$1.08	1	$1.08
Voltage Regulator	LM340T-5.0	$0.99	1	$0.99
Switch	EG1903	$0.71	1	$0.71
Battery Strap	2240k	$0.49	1	$0.49
Battery	P145	$2.13	1	$2.13
Capacitors (.1µF)	399-2054	$0.157	3	$0.471
Resistors (100 ?)	BC100YCT	$0.19	6	$1.14
Oscillator	SE1711	$2.94	1	$2.94
Voltage Level Shifter	Max233	$7.54	1	$7.54
9-Pin DSUB Receptacle	89909-8000	$4.42	1	$4.42
LED (blue)	L14006	$1.89	5	$9.45
Diode	Varies	~$0.00	1	$0.00
Resistor (330 ?)	OF331J	$0.54	1	$0.54
Resistor (220 ?)	OD221J	$0.42	1	$0.42
Capacitor (8.2 µF)	C1210C825K4PACTU	$0.396	1	$0.396
Microcontroller	Atmel Mega32	$8.00	1	$8.00
			Total:	$40.72

A glance at the above listing immediately shows that the cost of this system is quite low relative to those of alternative methods of creating musical annotation with standard MIDI devices. Thus the hardware portion of this project fulfills the goal of creating a fully functional and relatively inexpensive interface between a personal computer and MIDI device.

Firmware

Firmware for this system includes all code written for the Atmel Mega32 microcontroller. This code is written in AVR Code Vision C, compiled with the AVR Code Vision compiler, and manages the hardware interface between the MIDI device and the personal computer. The firmware is separated into three sub-sections, the first controls the software UART and serial interface, the second receives and filters MIDI data, and the third synchronizes events between the MIDI and serial interfaces.

Software UART

As the Mega32 offers only one on-chip UART, it is necessary to design a software UART in order to simultaneously maintain two serial connections. The serial interface to the PC operates using a half-duplex interrupt-driven UART created in software. This UART consists of send and receive buffers and a status byte which indicates whether the UART is busy, transmitting, and receiving. An RS232 line idles at logic high and then falls to indicate the start of a data transmission, so an external interrupt triggered on the falling edge of a port pin is a natural choice for implementation of the receive portion of the UART. When this external interrupt is triggered, a timer is started

according to the baud rate of the connection and the individual bits of a data byte are masked and shifted into the receive buffer. When the receive operation completes, the timer turns off and the UART idles. It should be noted that send operations are disabled during the full course of a receive operation, and the external interrupt which triggers a receive operation is disabled during receipt of a data byte, thus removing the possibility of concurrent receipt attempts. The send operation functions in a manner which is quite similar to that of the receive operation. When transmission of data is requested, a timer is again started in order to schedule the masking and shifting of individual bits. Requests for transmission are ignored during a transmit operation and the external interrupt which triggers a receive operation is disabled.

Timer0, an 8-bit timer and counter on the Mega32 is used for data manipulation. The microcontroller's clock runs at 16 MHz, and the timer used to schedule transmission and receipt of data is pre-scaled by 8, thus running at 2 MHz. The timer is preloaded with a value of 48 and is scheduled to interrupt upon overflow. The 8-bit register overflows at a value of 256. Therefore, an overflow interrupt occurs ever 212 ticks, or roughly every .106 milliseconds. This corresponds to a baud rate of almost exactly 9600 bits per second. At the start of a receive operation, the timer is set to interrupt after .159 milliseconds. This is because the external interrupt catches the falling edge of the port pin and thus the beginning of the data byte's start bit. Data values are best sampled towards the middle of a given bit and therefore it is necessary to wait for1.5 bits to pass upon receiving the external interrupt. It should also be noted that the initial preloaded timer values for the send and receive operations differ. This is to compensate

for the difference in the nature of the two operations. When a send operation is requested, the send algorithm simply begins to transmit data. However, a receive operation requires entry into and reentry from an external interrupt, a task which takes over 150 cycles when written in C. Clever use of assembly language could be employed to circumvent this difference; however, the implementation in C functions correctly and therefore is sufficient.

Several LED's indicate the status of the UART for the user. Specifically, LED0 is lit when a UART receive error is encountered, generally due to a mismatched baud rate. LED1 lights to signal that the UART is busy and that too many transmit or receive operations have been attempted simultaneously. LED2 lights at the beginning of a send operation and turns off when the operation completes, and LED3 behaves similarly for receive operations. The transmit pin for the UART is defined to be pin 3 of PORT D, and the receive pin is located at pin 2 of PORT D, the location of the Mega32's external interrupt 0. These LED's are connected to pins 0 through 3 of PORT B.

MIDI Interface

The protocol for interaction with a MIDI device is straightforward. MIDI data is sent asynchronously at a rate of 31.25kbaud. MIDI devices interact via "MIDI messages" which are multiple-byte packets conveying information about musical events. Although a MIDI device may generate hundreds of packets at a time, it is often the case that many of these packets contain superfluous information and can be ignored. For the purposes of this project, the microcontroller discards all messages expect those which fall within

a particular category of MIDI messages, the status messages. As a note is sounded on a MIDI device, a message is sent through the device's MIDI output port, indicating the occurrence of the event. This group of bytes has a fixed length and adheres to the following protocol: The first byte is always a status byte. This type of byte is the only MIDI message byte with the eighth bit set, and therefore values for this byte can range from 0x80 to 0xFF. The upper nibble of the byte determines the type of status byte:

Value	Meaning
8	Note Off
9	Note On
A	Aftertouch
B	Control Change
C	Program Change
D	Channel Pressure
E	Pitch Wheel

The values of interest for this project are 8 and 9, *Note Off* and *Note On*, respectively. These bytes indicate the start and stop of a note being sounded by a user. The lower nibble of the status byte indicates the channel of the event, which for this project is always channel 0. The first byte to follow a *Note On* or *Note Off* status byte indicates the note number of the key being pressed or released.

MIDI devices number notes within a range of 0 to 127, with “middle C” on a piano holding the place of number 60. The second byte to follow a *Note On* or *Note Off* status byte is a velocity byte. This byte contains information regarding the pressure on the particular key being depressed and is relevant only for devices with touch sensitivity support. The keyboard used to demonstrate this system is a Casio CTK-491 5-octave keyboard and does not support touch sensitive keys; therefore the velocity byte is not generally processed in the case of a *Note On* message. This byte was however used for *Note Off* messages. Certain keyboards substitute a *Note On* message with a velocity of 0 in lieu of a *Note Off* message (with an arbitrary velocity), and thus a *Note On* message must be checked in case it truly indicates a *Note Off* event.

The MIDI device is connected to the Mega32 microcontroller via the on-chip UART. The UART is set to interrupt upon receipt of a byte, and all processing code for MIDI data occurs within the corresponding interrupt service routine. The code to process MIDI data is sufficiently short and does not interfere with proper timing operation of the interrupt. A simple state machine processes each byte as it arrives, storing the note number of each note played. A timer runs in the background, interrupting once per millisecond. This is accomplished by pre-scaling timer1 by 64, creating a .4 microsecond time base. The timer is set to interrupt upon a compare match with the value 250, and thus a timer1 overflows interrupt is generated every 1 millisecond. This timer serves two purposes. It creates a time base with which to determine the duration of the sounding of a particular note, and it allows for the generation of a “heartbeat” LED. The timer inverts the voltage applied to pin 7 on PORT B every 500 ticks, thus

creating a heartbeat effect with the LED connected to this pin. This is useful for debugging purposes, especially when the microcontroller is not mounted on a development board, as it can serve as an indication that the processor is currently running. When a *Note On* message is received by the UART, the current value of the millisecond counter is stored. Upon receipt of the corresponding *Note Off* message, the stored start time of the note is subtracted from the current time, yielding the note's duration in milliseconds. The variables which store the current millisecond count, note start time, and note duration are integer types which can reach a maximum value of 65,535. For convenience, when the value of the current millisecond count reaches 65,000, it returns to 0. Because of this, it is possible that the counter may "roll over" between the depression and release of a key, generating a negative difference between the start time and end time. However, it is a straightforward procedure to check for this case and compensate accordingly. It is also possible for overflow to occur, in a situation where a user depresses a key for longer than 65 seconds. However, this case is extremely unlikely, and in the interest of saving computing cycles and memory space in the common case, the solution of using a larger storage structure to maintain a note's length was not employed. As MIDI data is received, information pertaining to each note is stored in a buffer and a count of packets present in the buffer is updated. However, the forwarding of MIDI data is left to the main loop of the program.

Process Synchronization

The main program loop and entry point in to the application is contained in a file entitled `SheetMusicGenerator.c`. This file includes two functions, *initialize* and

main. The *initialize* function makes calls to the individual initialization routines for both the software UART and the MIDI interface, and then globally unmasks interrupts. The *main* function calls *initialize* to set up all components of the firmware, and then enters an endless processing loop. Within this loop, the count of stored data packets is continuously checked. Upon discovery of a non-zero value, a note message is transmitted serially and the buffer of stored packets is updated accordingly. The order of operations for this sequence is as such in order to reduce possible conflict due to global variables shared between the main application loop and the MIDI interrupt. This is done in an effort to mimic the effects of an atomic test-and-set operation on the global index into the data buffer.

Each note depressed causes a message of 4 bytes to be sent to the personal computer. The first byte indicates the note's MIDI number, or pitch. The second two bytes are used to convey information about the duration of the note. Two bytes are necessary for this information as the software UART operates on 8-bit values and the note length value is stored in a 16-bit variable. Finally, a comma separator is sent to indicate the completion of one note message.

Code Organization

The code for the project firmware is organized into several files. `UART.h` contains the type definitions, function prototypes, and constant declarations for `UART.c` which implements the software UART for the RS232 serial interface to the personal computer. Similarly, `MIDI.h` contains definitions for `MIDI.c` which implements the interface to

the MIDI device. Finally, `SheetMusicGenerator.h` incorporates the MIDI and UART definitions into the file responsible for coordinating all events, `SheetMusicGenerator.c`. A full listing of the firmware code can be found in Appendix C.

Software

The software segment of this application is written in Microsoft Visual C++ using Microsoft Foundation Classes to implement a graphical user interface. A primary task for this part of the project was the selection of an appropriate language and an accompanying graphics design tool, followed by the mastery of the chosen language. Consideration was given to a variety of methods for creating a graphical user interface and MFC proved to be the most appropriate. Other possibilities included the use of C in conjunction with OpenGL or Java with Swing. A quick survey of OpenGL revealed that the tool is quite powerful and would allow for an interesting learning experience. However, the graphics rendering capabilities of OpenGL far exceed what is necessary for the construction of a simple graphical user interface. Because the tool is more suited for extensive, in-depth graphics rendering, it became apparent that the complexities which naturally accompany such a powerful tool could potentially hinder the learning and use of this product. The second language to be considered, Java, provided a much more appropriate set of graphics tools, the Java Swing package from Sun Microsystems. A significant portion of time was devoted to the study of both Java and Swing, providing a useful first insight into the field of object oriented programming. Finally, the notion of Java was dismissed in favor of a language with programmer controlled

memory management, C++. The time spent on the study of Java proved to be time well spent, as intuition about object oriented programming is an invaluable boost to the C++ learning curve. Extensive consideration and research determined that the C++ language with the enhancement of Microsoft Foundation Classes was perfectly suited to the needs of this project.

The primary purpose of the software component of this system is to display the results of interactions with a MIDI device, a task which is twofold. A connection must be established with the hardware via a COM port and any data transmitted along this connection must be displayed graphically to the user. With the introduction of a user interface comes the need for file manipulation. It can be assumed that a user will need to save the data he or she collects from the MIDI device for viewing and editing at a later date. Each of these tasks, along with the relevant sections of code, will be discussed in turn.

As discussed above, the Microsoft Foundation Classes are used to display all graphics for this application. The application's user interface is intuitive to use and conforms to the standards set forth by the Microsoft Developers Network, MSDN, for windows applications. The program consists of a single window, which is used to display any musical notation recorded, and to hold a set of menus through which the user interacts with the application. A screen capture of the running application follows.

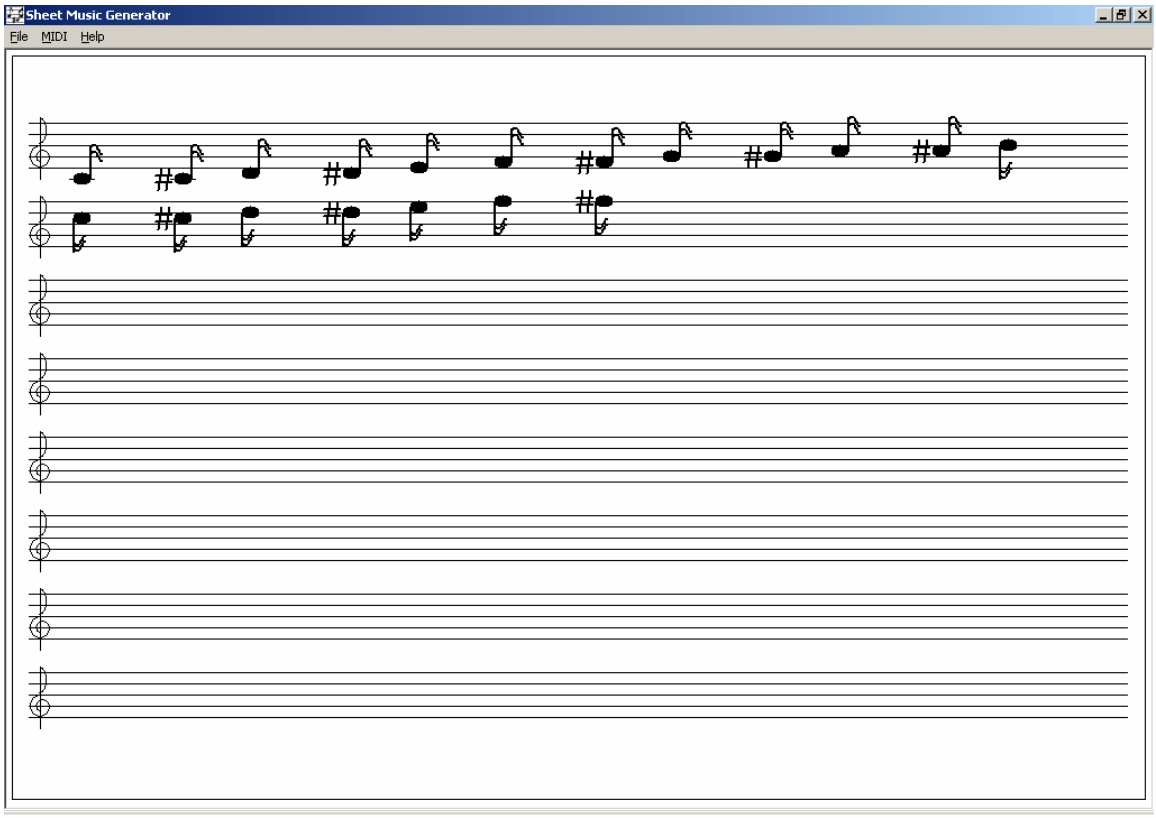


Figure 3: Software Application

The menus and other display items are contained in a text-editable resource file, which specifies names and locations for all controls. This file is entitled `AppResources.rc` and can be found in Appendix D. Corresponding definitions for constants used in this file appear in `ResourceIds.h`. The remainder of the implementation lies in the use of the classes provided by MFC.

An MFC program consists of two main parts, a main window object derived from the class CFrameWnd, and a single application object, derived from the class CWinApp. An instantiation of the application object begins the execution of the MFC program, prompting the application object's overloaded function, *InitInstance*, to create the main

window object and prepare the window to exchange messages with the Microsoft Windows operating system. The main window definition includes a section of code defining a message map. This map determines which messages sent to the application by the operating system are handled and which messages are ignored. It matches a handling function to each type of message specified by the programmer, thus allowing the programmer to create event driven code for each control in the application. The messages handled in this application are primarily those generated by user clicks on the various menu items. The program includes a *File* menu, with full support for opening, closing, and saving files. Interaction with the keyboard or other MIDI device is accomplished with the *MIDI* menu. This menu allows for the opening and closing of a connection to the MIDI device. Finally, the *Help* menu choice is used to display information about the application itself. The implementations for the main window and application objects are located in the file `Implementation.cpp`, and relevant class definitions can be found in the C++ header file `ClassSkeletons.h`.

As the user captures input from the MIDI device, results are displayed onscreen in the form of traditional sheet music. MFC provides an extensive list of classes for drawing directly to the screen, outside of the confines of control objects such as a buttons or textboxes. These classes and routines are used to render musical notes onscreen and thereby generate sheet music for the user. A musical score is an object oriented representation of one piece of sheet music, or one file. A score object contains a private linked list of note objects as well as a flag which indicates whether the score has been edited since it was last saved. The object also contains public routines for saving a

score, loading a previously saved score, drawing a score in the main window, adding a note object to a score, and determining whether a score has been edited (and therefore whether the user should be prompted to save before exiting). The most important members of the score object are its notes. A musical composition consists of a series of notes, each with a different pitch and length, and this concept is encapsulated in the application within the note object. The note object's private members include the note length and pitch, a pointer to the next note in a linked list, and routines for drawing specific pieces of a given note, such as its tail. A note's public methods allow other objects to retrieve or set the pointer to the next note in a linked list, determine parameters such as the note pitch and length, and draw the note onscreen. The final public member of the note object is a routine for drawing a musical sharp symbol before a note when appropriate. All objects within the program interact with the score object currently loaded onscreen, and the score object interacts with its individual note members. User input in the form of menu clicks prompts the main window to set or change the current score and the serial input connection updates the current score upon receipt of messages from a connected MIDI device.

A simple class for serial interaction obtained from www.codeguru.com is used for the low-level manipulation of the serial UART16550 port. The class includes functions to open and close a specific port with a given baud rate, and includes routines to read and write data to and from an opened port. The class is written in C++ and can be found in the files entitled `serial.h` and `serial.cpp` in Appendix D. The functions which

make use of the serial library are located in the file entitled `Implementation.cpp`. These functions manage the creation and use of a serial port object.

Oftentimes, a user will choose to open a connection with a MIDI device for data collection, but will expect to be able to simultaneously interact with other menu options within the application. Since MIDI data arrives asynchronously and may possibly arrive in clusters of packets, it is clear that the serial port must either be interrupt-driven or be read periodically until the user specifies that he or she has finished collecting data. It cannot simply be read once and then ignored. The serial class used is designed to work with the second of these methods, polling. Once the user indicates that he or she is ready to receive MIDI data, it is necessary that the serial port be polled periodically until further instruction from the user. This presents an interesting problem: code to read the serial port must run continuously, in effect monopolizing the computer's processor, but certain instructions from the user must be able to disrupt this polling procedure. Because user actions such as menu clicks are not interrupt driven in MFC, it is necessary that two separate threads of execution exist, one thread for polling the serial port and another for accepting user input. Therefore, when a user selects the *Capture* option from the menu, a serial connection is created and a new thread is spawned to retrieve and process data for this connection. When the user selects the *Close Connection* option, a flag is set by the main thread, notifying the serial thread of this action. The serial thread finishes its current operation and then closes cleanly.

Two global routines in the user interface remain to be discussed. The first routine, *SetTempo*, accepts an integer value of a desired tempo in beats per minute and calculates the length in milliseconds of each type of note present in musical notation. For instance, a setting of 120 beats per minute would imply that a quarter note receives .5 seconds; an eighth note receives .25 seconds, and so forth. As discussed above, the hardware sends to the software application a record of the length of time for which a particular note has been held. The second global function, *FindNoteLength* uses this value to determine which type of note to display. The implementations and class definitions for all note and score objects, as well as for the drawing of these objects, can be found in Appendix D, in the files entitled `Implementation.cpp` and `ClassSkeletons.h`.

The final aspect of the application to discuss is the implementation of file input/output routines. This program provides support for saving musical scores and for loading previously saved files. Interaction with the user for these tasks occurs through menu selections. Standard Microsoft Windows directory browsing is implemented using the common dialog objects provided by MFC. This allows the program to have a similar "look and feel" to that of most Microsoft Windows applications.

Results

This system succeeds in its goal to implement a fully functional interface between a MIDI device and a personal computer, and to graphically display MIDI data received on the computer. There are, however, certain limitations inherent in the design. One shortcoming is the possibility of overflow in the firmware variable which keeps track of the number of milliseconds for which a keyboard note is held down. Since the range of values for this variable is from 0 to 65,000, a user could cause overflow by holding a key down for longer than 65 seconds. This could clearly be fixed with the introduction of a larger storage container for the applicable variables, but such a solution necessitates the use of more memory and potentially more CPU cycles per operation. Therefore, the motto "make the common case fast" was followed in this design. Another drawback arises due to the conservative design of the software UART. In order to exclude the possibility of overlapping transmit operations, the firmware includes a delay after sending each serial byte. This then limits the speed at which data may be transported from the MIDI device to the computer, thus increasing the latency in the display when a user chooses to play notes quickly in succession. Again, the effects of this situation could be reduced if the duration of the delay were to be minimized, but such a change could cause the transmission of MIIDI data to become less reliable. Finally, it is possible that synchronization problems could appear due to the fact that the main firmware loop shares an index into a buffer with another routine. This is because the requests to read and subsequently update this variable are not guaranteed to operate

atomically. However, this situation is highly unlikely to arise and has not been encountered during a demonstration thus forth.

Despite these few drawbacks, the system performs quite well overall. Data is reliably transported from the MIDI device to the personal computer, and is processed by firmware in such a manner that it can be used efficiently by the software part of the system. The graphical user interface is functional, intuitive, and includes all of the expected and necessary features for MIDI data collection.

Conclusions

This project set out to address the task of allowing a musician to create musical scores on a personal computer using a digital musical instrument. This task is accomplished successfully by the system presented here. The system created relies on the use of a microcontroller, thus significantly reducing compatibility issues. Such a reliance on firmware also has the effect of dramatically reducing the cost of such a solution. Therefore, the system presented fully satisfies the goals specified at the start of this project.

Additionally, the project provided an equally important benefit in the area of education. Completion of this project mandated the mastery of several computer languages and called for extensive research and work in the areas of firmware and hardware. Serial communication protocols were explored from the point of view of both a personal computer and a small microcontroller. Finally, this project necessitated a thorough study into the creation of graphical user interfaces with the various tools available to developers today.

References

Azelson, Jan. Serial Port Complete. Madison, WIL Lake View Research, 2000.

Barnett, Cox, and O'Cull. Embedded C Programming and the Atmel AVR. Clifton Park, NY: Delmar Learning, 2003.

Kernighan, Brian W. and Dennis M. Ritchie. The C Programming Language, Second Ed. Marry Hill, NJ: AT&T Bell Laboratories 1988.

Schildt, Herbert. MFC programming from the ground up. Berkeley, CA: Osborne/McGraw-Hill, 1998.

Land, Dr. Bruce. ECE 476 Lecture Notes, Spring 2003,2004. Ithaca, NY: Dr. Bruce Land, 2003,2004.

http://www.borg.com/~jglatt/tech/midispec/intro.htm (Midi specification)

http://www.cplusplus.com/doc/tutorial/ (C++)

http://www.intap.net/~drw/cpp/ (C++)

http://www.codeguru.com/Cpp/I-N/network/serialcommunications/article.php/c2503/ (Serial Library)

http://msdn.microsoft.com/ (MFC and Visual C++ reference)

http://www.nbb.cornell.edu/neurobio/land/ (Serial reference)

DataSheets

- http://www.national.com/ds/LM/LM340.pdf (Voltage Regulator)
- http://www.toshiba.com/taec/components/Datasheet/6N138DS.pdf (Optoisolator)

- http://rocky.digikey.com/WebLib/E-Switch/Web%20Data/EG1201-1302%20Slide%20Switches.pdf (Switch)
- http://www.keyelco.com/kec/pdfs/p23.pdf (Battery Strap)
- http://rocky.digikey.com/WebLib/Panasonic/Web%20data/Panasonic_Alkaline_Hdbk_03-04_v1.pdf (Battery)
- http://rocky.digikey.com/WebLib/Kemet%20Caps/Web%20Data/Ceramic%20Conformally%20Coated%20-%20Radial%20Series.pdf (.1µF Capacitor)
- http://rocky.digikey.com/WebLib/BC%20Components/Web%20Data/5033E,5043E,5053H%20Metal%20Film%20Res.pdf (100 ? Resistor)
- http://rocky.digikey.com/WebLib/YAGEO/Web%20Data/MFR%20Series.pdf (100 k? Resistor)
- :http://rocky.digikey.com/WebLib/Epson/Web%20Data/SG-51,531%20Series.pdf (Oscillator)
- http://rocky.digikey.com/WebLib/Chicago%20Miniature/Web%20Data/Blue%20LED%20Lamps.pdf (Led)
- http://instruct1.cit.cornell.edu/courses/ee476/AtmelStuff/full32.pdf (Mega32)
- http://rocky.digikey.com/WebLib/3M/Web%20Data/899%20Series.pdf (D-SUB 9)
- http://rocky.digikey.com/scripts/ProductInfo.dll?Site=US&V=175&M=MAX233CPP (MAX233)

Appendices

Appendix A: Software User's Manual

Appendix B: Pictures and Schematics of Hardware

Appendix C: Firmware Code Listing

Appendix D: Software Code Listing

Appendix A: Software User's Manual

Use of the software application accompanying this system is quite straightforward. The user interface components of the application conform to the standards presented by the Microsoft Developer Network for Microsoft Windows applications. A view of the running application is depicted below.

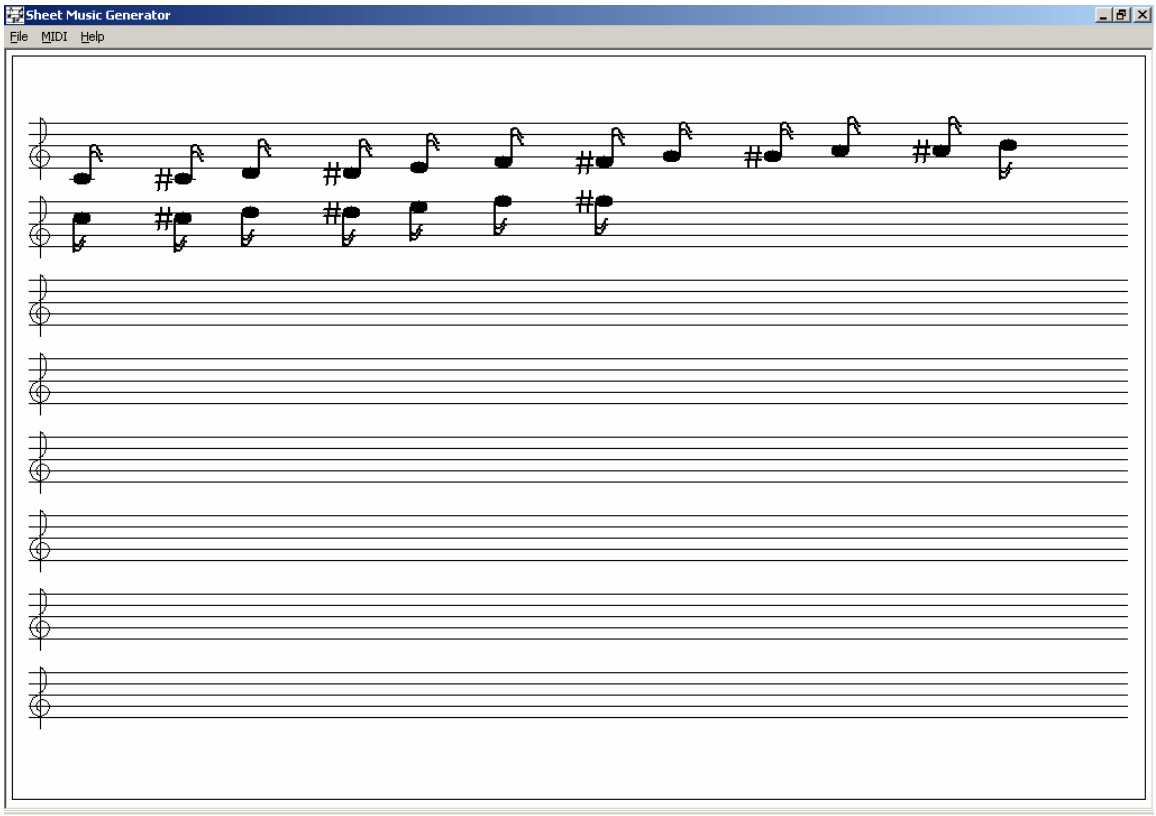


Figure 1: Software Application

File interaction is accomplished via the *File* menu. The *New* option creates a new score, the *Save* option saves the current score, if applicable, the *Open* option opens an existing score, and the *Close* option closes the current score. If the user attempts to close the current score, either by selecting *Close* from the *File* menu or by exiting the

program, creating a new score, or opening a different score, the application determines whether the current score needs to be save and prompts the user accordingly. The user may exit the application by selecting *Exit* from the *File* menu or by clicking the button in the top right corner of the window.

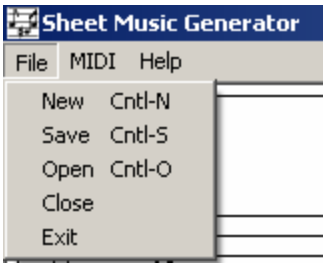


Figure 2: 'File' Menu

Interaction with the MIDI device is accomplished via the *MIDI* menu. The MIDI device must be attached to the system hardware using the MIDI cable provided. The serial cable extending from the hardware should be attached to the COM1 port of the personal computer. Finally, the power switch for the hardware should be moved to the 'on' position. When ready to collect data from the MIDI device, the user may select the *Capture* option from the *MIDI* menu in order to open a serial connection. When finished, the user may select the *Close Connection* option from the same menu.

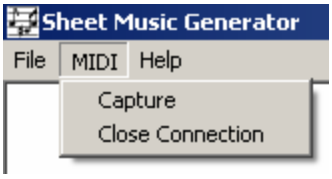
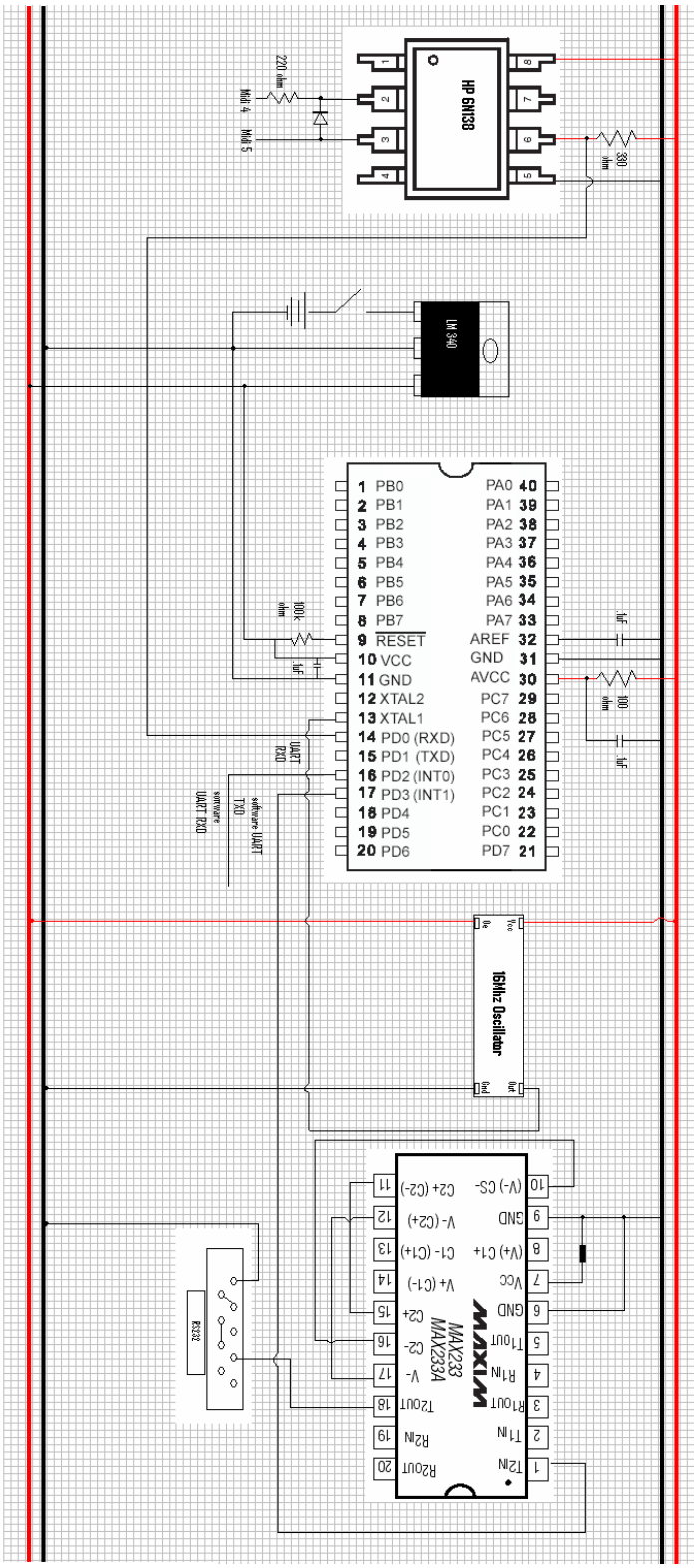


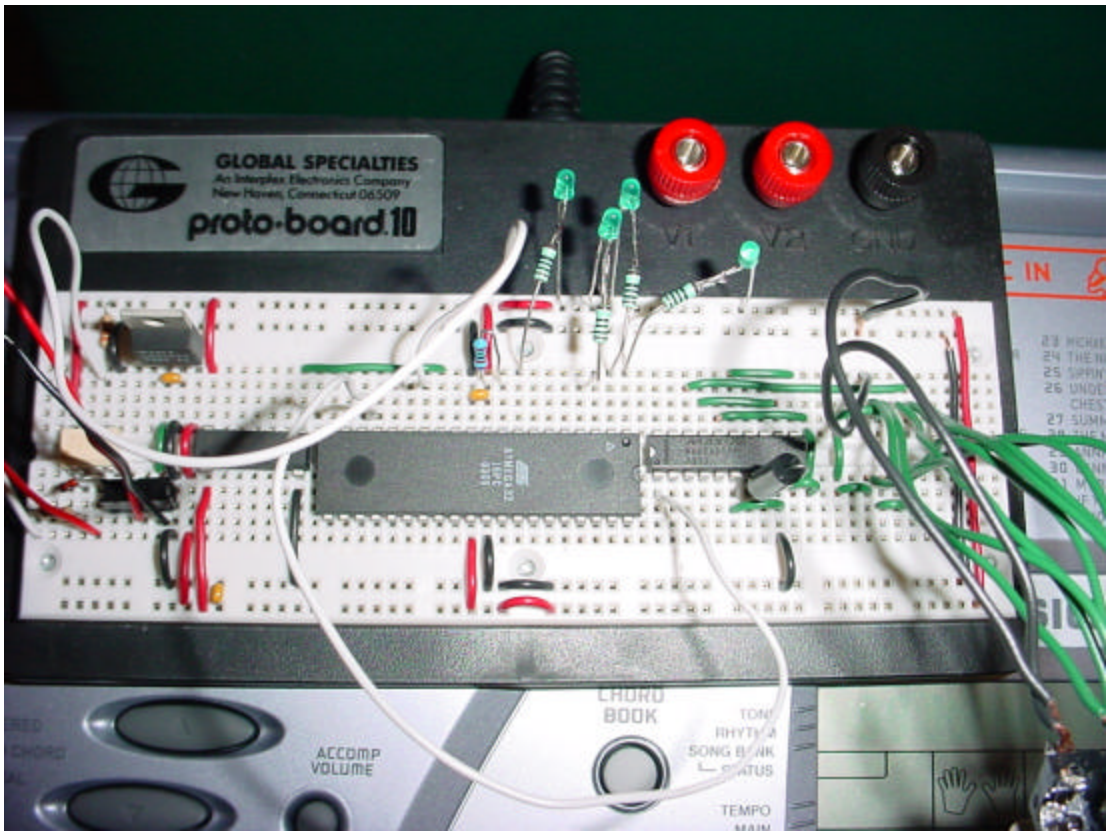
Figure 3: 'MIDI' Menu

Finally, the user may view information about the application by selecting the *About* option if the *Help* menu.

Figure 4: 'About' Menu

Appendix B: Pictures and Schematics of Hardware




GLOBAL SPECIALTIES
An Interplex Electronics Company
New Haven, Connecticut 06509
proto-board 10
ACCOMP VOLUME
CHORD BOOK
TEMPO


CHORD BOOK
TRANSPOSE / TUNE / PITCH
TONE
RHYTHM
SONG BANK

Appendix C: Firmware Code Listing

```
/*************************************************************************************************************
Meg Walraed-Sullivan
Cornell University M.Eng Project
Submitted: August, 2004

This file contains the prototypes and definitions for the entire project
*************************************************************************************************************/


/*-------------------------------------------------------------------------------------------
Includes
-------------------------------------------------------------------------------------------*/

#include <Mega32.h>                                             //microcontroller defines, etc
#include "uart.h"                                               //software UART stuff
#include "MIDI.h"                                               //MIDI interface


/*-------------------------------------------------------------------------------------------
Defines
-------------------------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------------------------
Prototypes
-------------------------------------------------------------------------------------------*/
void initialize(void);                                          //initialize all components

/*************************************************************************************************************
Meg Walraed-Sullivan
Cornell University M.Eng Project
Submitted: August, 2004

This file contains the implementation for a half-duplex interrupt-driven software UART
*************************************************************************************************************/


/*-------------------------------------------------------------------------------------------
Includes
-------------------------------------------------------------------------------------------*/

#include "SheetMusicGen.h"                                      //prototypes, defines, etc
#include "delay.h"

/*-------------------------------------------------------------------------------------------
Globals
-------------------------------------------------------------------------------------------*/




/*-------------------------------------------------------------------------------------------
Implementation
-------------------------------------------------------------------------------------------*/



/*************************************************************************************************************
function:   main
description:        entry point and task loop for program
args:               none
returns:    none
*************************************************************************************************************/
```

```
void main(void){

        char low_byte;
        char high_byte;

        initialize();                                   //initialize all components

        while(1){


            if (new_data_packets>0){                    //if there is MIDI data

                begin_xmit(midi_in[first_packet_index]);    //send first packet
                delay_ms(100);

                low_byte = (char) (0x00FF &  len_in[first_packet_index]);
                high_byte = (char)((len_in[first_packet_index]&0xFF00)>>8);

                   begin_xmit(high_byte);               //send second  packet
                delay_ms(100);

                begin_xmit(low_byte);                   //send third packet
                delay_ms(100);

                begin_xmit(',');                        //send separator
                delay_ms(100);

                first_packet_index++;                   //move to next packet

                if (first_packet_index==max_MIDI_packets)   //rollover index if necessary
                        first_packet_index++;

                new_data_packets--;                     //decrement the number of packets

            }//if (new_data_packets>0)

        }//while

 }//main

 /*****************************************************************************************************
function:  initialize
description:    initializes all components and then starts interrupts
args:           none
returns:    none
*****************************************************************************************************/
void initialize(void){

        UART_initialize();                              //initialize the UART component
        MIDI_initialize();                              //initialize the MIDI interface

        #asm("sei");                                    //turn on interrupts

 }//initialize

/*****************************************************************************************************
Meg Walraed-Sullivan
Cornell University M.Eng Project
Submitted: August, 2004

This file contains the prototypes and definitions for a half-duplex interrupt-driven software UART
*****************************************************************************************************/


/*---------------------------------------------------------------------------------------
Includes
---------------------------------------------------------------------------------------*/

#include <Mega32.h>                                     //microcontroller defines, etc
```

```
#include "general.h"                                    //software UART stuff

/*-------------------------------------------------------------------------------------------
Defines
-------------------------------------------------------------------------------------------*/
#define UART_busy_mask 0b00000001                       //masks for UART status byte, indicate
#define UART_xmit_mask 0b00000010                       //whether UART is busy, transmitting
#define UART_data_mask 0b00000100                       //or has data to receive
#define UART_busy_mask_inverse 0b11111110
#define UART_xmit_mask_inverse 0b11111101
#define UART_data_mask_inverse 0b11111011

#define UART_xmit PORTD.3                               //define PORT pins used for UART
#define UART_rcv PIND.2
#define UART_error_led      PORTB.0
#define UART_busy_led       PORTB.1
#define UART_send_led       PORTB.2
#define UART_rcv_led        PORTB.3


/*-------------------------------------------------------------------------------------------
Prototypes
-------------------------------------------------------------------------------------------*/
void UART_initialize(void);                             //initialize everything necessary
void begin_xmit(char byte_to_send);                     //sets up UART for sending
void begin_rcv(void);                                   //sets up UART for receiving
void finish_xmit(void);                                 //cleans up after a send
void finish_rcv(void);                                  //cleans up after a receive
void enable_UART_timer_send(void);                      //turns on timer for UART send
void enable_UART_timer_rcv(void);                       //turns on timer for UART receive



/*****************************************************************************************************
Meg Walraed-Sullivan
Cornell University M.Eng Project
Submitted: August, 2004

This file contains the implementation for a half-duplex interrupt-driven software UART
*****************************************************************************************************/


/*-------------------------------------------------------------------------------------------
Includes
-------------------------------------------------------------------------------------------*/

#include "UART.h"                                       //prototypes, defines, etc


/*-------------------------------------------------------------------------------------------
Globals
-------------------------------------------------------------------------------------------*/

char UART_status_byte;                                  //status byte for software UART
char UART_send_buffer;                                  //holds byte to send on UART
char UART_rcv_buffer;                                   /holds byte received on UART

char UART_bits_done;                                    //number bits xmitted/received
char UART_bits_done_mask;                               //mask for sending individual bits



/*-------------------------------------------------------------------------------------------
Implementation
-------------------------------------------------------------------------------------------*/
```

```
/*****************************************************************************************************
function:   timer0 overflow interrupt
description:        resets the timer to interrupt again in 52 us
args:               none
returns:    none
*****************************************************************************************************/
interrupt [TIM0_OVF] void timer0_overflow(void){

        TCNT0=48;                                               //reload to interrupt again in 52us

        if (UART_status_byte & UART_busy_mask) {                //UART is xmitting/receiving

                if (UART_status_byte & UART_xmit_mask){         //UART is transmitting

                        switch (UART_bits_done){                //see how many bits sent so far

                                case 0:                                 //first bit is start bit
                                        UART_xmit=PORT_lo;      //start bit is low
                                        break;

                                case 9:                                 //last bit is stop bit
                                        UART_xmit=PORT_hi;      //stop bit is high
                                        break;

                                case 10:                                //first idle bit
                                        UART_xmit=PORT_hi;      //idle high
                                        break;

                                case 11:                                //first idle bit
                                        UART_xmit=PORT_hi;      //idle high
                                        finish_xmit();          //clean up

                                        break;

                                default:                                //data bit
                                        if (UART_send_buffer&UART_bits_done_mask)
                                                UART_xmit=PORT_hi;
                                        else
                                                UART_xmit=PORT_lo ;

                                         UART_bits_done_mask<<=1;   //move mask to next bit

                        }//switch (UART_bits_done)

                        UART_bits_done++;                       //increment # bits sent

                }//if (UART_status_byte & UART_xmit_mask)
                else {

                        switch (UART_bits_done){                //see how many bits received so far

                                case 0:                                 //no bits sampled yet
                                        if(UART_rcv==PORT_hi)   //start bit should be low
                                                UART_error_led=led_on;
                                        UART_bits_done++;  //middle of first data bit
                                        break;

                                case 9:                                 //last bit should be stop bit
                                        if (UART_rcv==PORT_lo)  //if stop bit is lo, problem
                                                UART_error_led = led_on;

                                        finish_rcv();           //clean up
                                        break;

                                default:                                //data bit

                                        if(UART_rcv==PORT_hi)   //if received a 1, set bit
                                                UART_rcv_buffer |= UART_bits_done_mask;
```

```
                                        UART_bits_done++;               //increment # bits received

                                        UART_bits_done_mask<<=1;    //move mask to next bit

                                }//switch (UART_bits_done)

                        }//else

                }//if (software_UART_status_byte&software_UART_busy_mask)

}//timer0_overflow


/*****************************************************************************************************
function:  external interrupt 0
description:            indicates that UART has something to receive
args:                   none
returns:    none
*****************************************************************************************************/
interrupt [EXT_INT0] void external_interrupt0(void){

        if (!(UART_status_byte & UART_data_mask))
                begin_rcv();                                    //start receiving!

}//external_interrupt0



/*****************************************************************************************************
function:  UART_initialize
description:            sets up software UART
args:                   none
returns:    none
*****************************************************************************************************/
void UART_initialize(void) {


        MCUCR |=  0b00000010;                                   //external interrupt0 occurs on falling edge
        GICR |=   0b01000000;                                   //unmask external interrupt0

        UART_status_byte = 0b00000000;                          //no data ready, not busy/transmitting
        UART_send_buffer=0;                                     //no bytes to send yet
        UART_rcv_buffer=0;                                      //no bytes received yet

        UART_bits_done = 0;                                     //no bits to xmit/receive
        UART_bits_done_mask=0b00000001;                         //start with first bit

        DDRB = 0xFF;                                            //PORTB used as indicator LEDS

         PORTB = 0xFF;                                          //start with all the leds off

        DDRD = 0b00001000;                                      //PORTD inputs except UART xmit
        PORTD = 0xFF;                                           //turn on pullups, idle xmit hi

}//UART_initialize

/*****************************************************************************************************
function:  begin_xmit
description:            sets up UART for a send
args:                   (char) data to send with the software UART
returns:    none
*****************************************************************************************************/
void begin_xmit(char byte_to_send){

        if (UART_status_byte & UART_busy_mask)                  //if UART is busy
                UART_busy_led = led_on;                         //report an error to user

        else{                                                   //if UART not busy
```

```
                UART_send_led = led_on;                         //report send action to user

                GICR &=   0b10111111;                           //turn off external interrupt0

                UART_status_byte |= UART_busy_mask;             //set busy bit
                UART_status_byte |= UART_xmit_mask;             //set transmitting bit

                UART_send_buffer=byte_to_send;                  //set byte to send

                UART_bits_done = 0;                             //no bits sent yet
                UART_bits_done_mask=0b00000001;                 //start with first bit

                enable_UART_timer_send();                       //start the UART timer

        }//else

}//begin_xmit

/*****************************************************************************************************
function:  begin_rcv
description:    sets up UART for a receive
args:           none
returns:    none
*****************************************************************************************************/
void begin_rcv(void){

        if (UART_st atus_byte & UART_busy_mask)                 //if UART is busy
                UART_busy_led = led_on;                         //report an error to user

        else{                                                   //if UART not busy

                UART_rcv_led = led_on;                          //report receive action to user

                GICR &=   0b10111111;                           //turn  off external interrupt0

                UART_status_byte |= UART_busy_mask;             //set busy bit
                UART_status_byte &= UART_xmit_mask_inverse;     //clear transmit bit

                UART_bits_done = 0;                             //no bits received yet
                UART_bits_done_mask=0b00000001;                 //start with first bit

                UART_rcv_buffer=0;                              //start with all 0 in receive buffer

                enable_UART_timer_rcv();                        //start the UART timer

        }//else

}//begin_rcv


/*****************************************************************************************************
function:  finish_xmit
description:    resets UART after a send
args:           none
returns:    none
*****************************************************************************************************/
void finish_xmit(void){

        TIMSK &= 0b11111110;                                    //stop the UART timer

        UART_status_byte &= UART_busy_mask_inverse;             //clear busy bit
        UART_status_byte &= UART_xmit_mask_inverse;             //clear transmitting bit


        UART_bits_done = 0;                                     //reset bits sent

        UART_send_led = led_off;                                //turn off the send led

        GICR |=   0b01000000;                                   //turn external interrupt0 back on
```

```
}//finish_xmit


/*************************************************************************************************
function:  finish_rcv
description:        resets UART after a receive
args:               none
returns:    none
*************************************************************************************************/
void finish_rcv(void){

        TIMSK &= 0b11111110;                                    //stop the UART timer

        UART_status_byte &= UART_busy_mask_inverse;             //clear busy bit
        UART_status_byte |= UART_data_mask;                     //set data ready bit

        UART_bits_done = 0;                                     //reset bits received

        PORTD.2=1;

        GICR |=  0b01000000;                                    //turn external interrupt0 back on

        UART_rcv_led = led_off;                                 //turn off the receive led

}//finish_rcv


/*************************************************************************************************
function:  enable_UART_timer_send
description:        sets up and starts the UART timer for sending
args:               none
returns:    none
*************************************************************************************************/
void enable_UART_timer_send(void){

        TCNT0 =  48;                                            //reload timer0 with 48
        TCCR0 =  0b00000010;                                    //set timer0 prescalar to 8
        TIMSK |=  0b00000001;                                   //unmask timer0 overflow interrupt

}//enable_UART_timer_send

/*************************************************************************************************
function:  enable_UART_timer_rcv
description:        sets up and starts the UART timer for receiving
args:               none
returns:    none
*************************************************************************************************/
void enable_UART_timer_rcv(void){

        TCNT0 =  180;                                           //reload timer0 with 48
        TCCR0 =  0b00000010;                                    //set timer0 prescalar to 8
        TIMSK |=  0b00000001;                                   //unmask timer0 overflow interrupt

}//enable_UART_timer_rcv

/*************************************************************************************************
Meg Walraed-Sullivan
Cornell University M.Eng Project
Submitted: August, 2004

This file contains the prototypes and definitions for an interface to a MIDI controller
*************************************************************************************************/


/*-------------------------------------------------------------------------------------------
Includes
-------------------------------------------------------------------------------------------*/
#include <Mega32.h>                                             //microcontroller defines, etc
#include "general.h"                                            //software UART stuff
```

```
/*-------------------------------------------------------------------------------------------
Defines
-------------------------------------------------------------------------------------------*/

#define MIDI_error_led PORTB.7                          //user interface to MIDI device
#define max_MIDI_packets 100                            //maximum number of pending packets
#define MIDI_status_mask 128                            //mask to check for status byte

/*-------------------------------------------------------------------------------------------
Prototypes
-------------------------------------------------------------------------------------------*/
void MIDI_initialize(void);                             //initialize everything necessary




/*****************************************************************************************************
Meg Walraed-Sullivan
Cornell University M.Eng Project
Submitted: August, 2004

This file contains the implementation for an interface to a MIDI controller
*****************************************************************************************************/


/*-------------------------------------------------------------------------------------------
Includes
-------------------------------------------------------------------------------------------*/

#include "MIDI.h"                                       //prototypes, defines, etc


/*-------------------------------------------------------------------------------------------
Globals
-------------------------------------------------------------------------------------------*/
char midi_in[max_MIDI_packets];                         //storage for midi data received
unsigned int len_in[max_MIDI_packets];                  //storage for midi lengths calculated
char new_data_packets;                                  //number of packets to send to pc
char packet_array_index;                                //index to array of data received
char first_packet_index;                                //index of first packet in array


char state;                                             //state of MIDI receiver
enum {      status_state,
            note_num_state,
            velocity_state,
            end_status_state,
            end_note_num_state,
            end_velocity_state};

unsigned int cur_note_start_ms;                         //records start time of notes
unsigned int ms_count;                                  //number of milliseconds that have passed

char curr_note_num;                                     //current note's number
unsigned int curr_note_ms_dff;                           //current note's length in ms

/*-------------------------------------------------------------------------------------------
Implementation
-------------------------------------------------------------------------------------------*/


/*****************************************************************************************************
function:   UART receive interrupt
description:        reads and handles data received from the UART
args:               none
```

```
returns:    none
*********************************************************************************************************/
interrupt [USART_RXC] void MIDI_data_received(void){

  char temp_UDR_storage;

  if (new_data_packets<=max_MIDI_packets){                          //if there is room in temp storage

        temp_UDR_storage = UDR;                                     //grab packet

        switch (state) {                                            //run state machine (inlined for speed)

              //waiting for first status byte
              case status_state:
             curr_note_ms_dff=0;
                    if (!(temp_UDR_storage&MIDI_status_mask))       //if didn't receive status byte
                      MIDI_error_led = led_on;                      //should've gotten status byte
                    state=note_num_state;                           //wait for note number
                    break;

              case note_num_state:

                    curr_note_num = temp_UDR_storage;               //record note number

                    cur_note_start_ms = ms_count;                   //record current time

                    state=velocity_state;
                    break;

              case velocity_state:
                    state=end_status_state;
                    break;

              case end_status_state:
                                                                    //calculate length of note
                    if (ms_count-cur_note_start_ms>0)               //if clock didn't roll during note
                          curr_note_ms_dff =  ms_count-cur_note_start_ms;
                    else                                            //if clock rolled, compensate
                          curr_note_ms_dff = (65000-cur_note_start_ms) + ms_count;

                    state=end_note_num_state;
                    break;

              case end_note_num_state:

                    midi_in[packet_array_index]=curr_note_num;      //save note number for transmitting
                    len_in[packet_array_index]=curr_note_ms_dff;    //save note number for transmitting
                    packet_array_index++;                           //move to next storage space
                    if (packet_array_index==max_MIDI_packets)       //roll over index if necessary
                          packet_array_index=0;
                    new_data_packets++;

                    state=end_velocity_state;
                    break;

              case end_velocity_state:
                   state=status_state;
                    break;
        }//switch

  }//if (new_data_packets<=max_MIDI_packets)

  else                                                              //if temp storage is full
         MIDI_error_led = led_on;                                   //alert user

}//MIDI_data_received


/*********************************************************************************************************
function:  tim1_cmpA
```

```
description:        occurs every 1ms, updates time
args:               none
returns:    none
*********************************************************************************************************/
interrupt [TIM1_COMPA] void tim1_cmpA(void){


        if (ms_count++%500==0)                          //increment the # of ms passed
                PORTB.7=~PORTB.7;                       //blink LED every 1/2 second

        if (ms_count==65000)                            //if a second has passed
                ms_count=0;                             //reset ms count

}//tim1_cmpA



/*********************************************************************************************************
function:   MIDI_initialize
description:        sets up interface to MIDI device
args:               none
returns:    none
*********************************************************************************************************/
void  MIDI_initialize(void) {

        UBRRL = 31;                                     //UBRR = f_osc/16BAUD -1
        UCSRB = 0b10010000;                             //enable receive and receive interrupt
        UCSRC = 0b00000000;                             //asynchronous, no parity, 1 stop bit

        new_data_packets=0;                             //nothing received yet
        packet_array_index=0;                           //start at beginning of the array
        first_packet_index=0;                           //first packet will be in beg. of array

        ms_count=0;                                     //start at time 0


        OCR1A = 250;                                    //250 ticks * 4us/tick = 1ms/interrupt
        TCCR1B = 0b00001011;                            //clear compare, prescale by 64
        TCCR1A = 0x00;
        TIMSK = 0b00010000;                             //unmask timer1 compare interrupt

        state=status_state;

}//MIDI_initialize
```

Appendix D: Software Code Listing

```
/********************************************************************************************

        Meg Walraed-Sullivan
        Cornell University School of Electrical Engineering
        Masters of Engineering Project: Sheet Music Generator

        Resources.RC

        This file includes all graphical resources used in the application

********************************************************************************************/

#include <ResourceIds.h>                                        //ids of all resources used in app
#include <afxres.h>                                             //dialogs, controls


/*------------------------------------------------------------------------------
Menu
------------------------------------------------------------------------------*/
MainMenu MENU {

        POPUP "&File" {

                MENUITEM "&New\tCntl-N", IDM_FILE_NEW
                MENUITEM "&Save\tCntl-S", IDM_FILE_SAVE
                MENUITEM "&Open\tCntl-O", IDM_FILE_OPEN
                MENUITEM "&Close", IDM_FILE_CLOSE
                MENUITEM "&Exit", IDM_FILE_EXIT

        }//POPUP "&File

        POPUP "&MIDI" {

                MENUITEM "&Capture", IDM_SCORE_KEY
                MENUITEM "&Close Connection", IDM_SCORE_STOP

        }//POPUP "&Score"

        POPUP "&Help" {

                MENUITEM "About", IDM_HELP_ABOUT

        }//POPUP "&Help"

}//MainMenu MENU


MainMenu ACCELERATORS {

        "^N", IDM_FILE_NEW
        "^S", IDM_FILE_SAVE
        "^O", IDM_FILE_OPEN

}//MainMenu ACCELERATORS

/*------------------------------------------------------------------------------
About Box
------------------------------------------------------------------------------*/

AboutDialog DIALOG 200, 100, 142, 92

CAPTION "About: Sheet Music Generator"
```

```
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION  {

        CTEXT "Sheet Music Generator\n\tVersion 1.0\n\nMeg Walraed-Sullivan", IDD_ABOUTTEXT, 20, 10, 100, 40

        PUSHBUTTON "Ok", IDOK, 52, 65, 37, 14,WS_CHILD | WS_VISIBLE | WS_TABSTOP

}//AboutDialog


/*-------------------------------------------------------------------------
Icon/Cursor
-------------------------------------------------------------------------*/
MainIcon ICON MUSIC.ICO

/*****************************************************************************

        Meg Walraed-Sullivan
        Cornell University School of Electrical Engineering
        Masters of Engineering Project: Sheet Music Generator

        ClassSkeleton.h

        This file includes all class definitions used in the application

*****************************************************************************/

#include "serial.h"                                             //serial interface


//note sizes
typedef enum{sixteenth_note,eighth_note,quarter_note,half_note,whole_note} note_length_t;

/*-------------------------------------------------------------------------
class CMainWin:
        Main Window of Application
-------------------------------------------------------------------------*/
class CMainWin :public CFrameWnd{

        CBitmap m_bmp;                                          //virtual window bitmap
        CBrush m_bkbrush;                                       //brush for virtual window

        CPen m_ScorePen;                                        //pen used to draw score
        CBrush m_ScoreBrush;                                    //brush used for score background

public:

        CWinThread *serial_thread;                              //thread for serial interaction
        CSerial serial_conn;                                    //serial connection object

        bool serial_thread_running;                             //indicates whether the serial thread exists
        bool serial_thread_stop_flag;                           //flag used to kill the serial thread


        CDC m_memDC;                                            //virtual window device context

        CMainWin(LPCSTR ClassName);                             //const ructor (includes style class)

        afx_msg void OnPaint();                                 //for displaying the window

        afx_msg void OnDestroy();                               //respond to closing the window

        afx_msg void  OnFileNew();                              //respond to menu clicks
        afx_msg void  OnFileSave();
        afx_msg void  OnFileOpen();
        afx_msg void  OnFileClose();
        afx_msg void  OnFileExit();

        afx_msg void  OnScoreKey();
        afx_msg void  OnScoreStop();
```

```
        afx_msg void  OnHelpAbout();

        void DrawBlankScore();                                  //draws a new blank score
        void DrawClef(int y_pos);                               //draws a clef at given height
        void ClearScoreScreen();                                //clears the drawing area
        void CheckSave();                                       //check for a score to save
        void DestroyCurrentScore();                             //destroys object holding the current score


        DECLARE_MESSAGE_MAP()                                   //message map

};//class CMainWin

/*--------------------------------------------------------------------------------
class CSerialThread:
        Thred which controls serial access
--------------------------------------------------------------------------------*/
class CSerialThread :public CWinThread{

public:

        CSerialThread();                                        //use default constructor

        DECLARE_MESSAGE_MAP()                                   //message map

};//CSerialThread

/*--------------------------------------------------------------------------------
class CAboutDialog:
        "About" Dialog Box
--------------------------------------------------------------------------------*/

class CAboutDialog: public CDialog{

public:

        CAboutDialog(char *DialogName, CWnd *Owner) :    //constructor
          CDialog(DialogName, Owner) {}                         //use parent's constructor

};//class CAboutDialog

/*--------------------------------------------------------------------------------
class CMainApp:
        Main Application Object
--------------------------------------------------------------------------------*/
class CMainApp:public CWinApp {

public:
        BOOL InitInstance();                                    //override InitInstance from CWinApp

};//class CMainApp

/*--------------------------------------------------------------------------------
class Note:
        Note Object
--------------------------------------------------------------------------------*/
class CNote{

        int note_num;                                           //pitch of note
        note_length_t note_length;                              //length of note

        CNote* next_note;                                       //pointer to the next note in list

        void DrawTail(int x_start, int y_start, CDC*);          //adds the tail(s) onto a note
        void DrawUpsideDownTail(int x_start, int y_start, CDC*);//adds the tail(s) onto a note

public:
        CNote();                                                //empty constructor
        CNote(int num,note_length_t length);                    //parameter constructor
```

```
	void DrawNote(int cur_x, int staff_bottom,CDC*);		//draws the note onscreen
	void DrawSharp(int start_x, int start_y, CDC*);		//draws a sharp sign

	void SetNext(CNote* next_n);		//sets next note in list
	CNote* GetNext(void);		//returns the next note
	int GetNoteNum();		//returns the pitch value
	note_length_t GetNoteLength();		//returns the length

};//class Note


/*---------------------------------------------------------------------------
class CScore:
	Score Object
---------------------------------------------------------------------------*/
class CScore{

	bool edited;
	//whether score has been edited since loaded
	CNote* note_list;		//linked list of notes
	CNote* last_note;		//last note in linked list

public:
	CScore();		//constructor
	~CScore();		//destructor

	void AddNote (int num,note_length_t len);		//adds a note to the score
	bool IsEdited();		//returns whether score has been edited
	bool SaveScore ();		//saves the current score to a file
	void LoadScore ();		//loads a score from a file

	void DrawScore(CMainWin*);		//displays the score onscreen

};//class CScore


/*****************************************************************************

	Meg Walraed-Sullivan
	Cornell University School of Electrical Engineering
	Masters of Engineering Project: Sheet Music Generator

	DrawingConstants.h

	This file includes all constants used for drawing.

*****************************************************************************/

//score borders
#define LEFT_BORDER 5
#define RIGHT_BORDER 5
#define TOP_BORDER 5
#define BOTTOM_BORDER 5
#define INNER_BORDER 15

#define SCORE_START_Y 2*BETWEEN_STAFFS+TOP_BORDER
#define SCORE_START_X LEFT_BORDER + INNER_BORDER + 40

//staff constants
#define BETWEEN_STAFFS 30
#define BETWEEN_LINES 10
#define STAFF_HEIGHT 4*BETWEEN_LINES

//clef constants
#define CLEF_TOP_OVERHANG 5
#define CLEF_BOTTOM_OVERHANG 10
#define CLEF_INDENT 10
#define CLEF_X CLEF_INDENT+RIGHT_BORDER+INNER_BORDER
#define CLEF_TOP_ARC_WIDTH 6
```

```
//note constants
#define NOTE_WIDTH 15
#define NOTE_HEIGHT BETWEEN_LINES
#define NOTE_STEM BETWEEN_LINES*3
#define BETWEEN_NOTES NOTE_WIDTH*4
#define TAIL_HEIGHT BETWEEN_LINES
#define TAIL_WIDTH TAIL_HEIGHT*3/4
#define TAIL_SPACING TAIL_HEIGHT/2
#define C_LINE_OVERHANG 4

//sharp/flat symbols
#define SHARP_WIDTH NOTE_WIDTH
#define SHARP_SKEW 3
#define SHARP_HEIGHT NOTE_HEIGHT+1
#define SHARP_OFF_LINE 3

//individual staff lines
#define LINE_LENGTH 900

/*****************************************************************************************

        Meg Walraed-Sullivan
        Cornell University School of Electrical Engineering
        Masters of Engineering Project: Sheet Music Generator

        FileConstants.h

        This file includes all constants used for file IO.

*****************************************************************************************/

#define FILE_BUFFER_LEN 100

#define FILE_DEFAULT_NAME "my_music.txt"

/*****************************************************************************************

        Meg Walraed-Sullivan
        Cornell University School of Electrical Engineering
        Masters of Engineering Project: Sheet Music Generator

        ResourceIds.h

        This file includes all ids of all resources used in the application.

*****************************************************************************************/


/*---------------------------------------------------------------------------
Menu:1000
---------------------------------------------------------------------------*/

//File Menu: 000
#define IDM_FILE_NEW        1000
#define IDM_FILE_SAVE       1001
#define IDM_FILE_OPEN       1002
#define IDM_FILE_CLOSE      1003
#define IDM_FILE_EXIT       1005

//Score Menu:100
#define IDM_SCORE_KEY 1100
#define IDM_SCORE_STOP 1101

//Help Menu:200
#define IDM_HELP_ABOUT 1200


/*---------------------------------------------------------------------------
Dialogs:2000
---------------------------------------------------------------------------*/
```

```
//About Box: 000
#define IDD_ABOUTTEXT 2000

// Serial.h

#ifndef __SERIAL_H__
#define __SERIAL_H__

#define FC_DTRDSR       0x01
#define FC_RTSCTS       0x02
#define FC_XONXOFF      0x04
#define ASCII_BEL       0x07
#define ASCII_BS        0x08
#define ASCII_LF        0x0A
#define ASCII_CR        0x0D
#define ASCII_XON       0x11
#define ASCII_XOFF      0x13

class CSerial
{

public:
        CSerial();
        ~CSerial();

        BOOL Open( int nPort = 2, int nBaud = 9600 );
        BOOL Close( void );

        int ReadData( void *, int );
        int SendData( const char *, int );
        int ReadDataWaiting( void );

        BOOL IsOpened( void ){ return( m_bOpened ); }

protected:
        BOOL WriteCommByte( unsigned char );

        HANDLE m_hIDComDev;
        OVERLAPPED m_OverlappedRead, m_OverlappedWrite;
        BOOL m_bOpened;

};

#endif

/************************************************************************************

        Meg Walraed-Sullivan
        Cornell University School of Electrical Engineering
        Masters of Engineering Project: Sheet Music Generator

        Implementation.cpp

        This file includes the implementations for each class in the application

************************************************************************************/


/*_____________________________________________________________________________

Includes
_____________________________________________________________________________*/

#include <afxwin.h>                                             //all MFC headers, classes, etc
#include <afxdlgs.h>                                            //common dialogs
#include <fstream.h>                                            //file IO
#include <afxmt.h>                                              //threads
```

```
#include "ClassSkeleton.h"                                  //class definitions
#include "ResourceIds.h"                                    //ids of all resources
#include "DrawingConstants.h"                               //constants used to draw score
#include "FileConstants.h"                                  //constants used to do File IO
//#include "serial.h"                                       included in class skeleton

/*_____________________________________________________________________________

Globals
_____________________________________________________________________________*/

int screen_width=0;                                         //width of entire window whem maximize
int screen_height=0;                                        //height of entire window whem maximize

CScore* curr_score=NULL;                                    //reference to the current score object


int quarter_note_ms;                                        //information about current tempo
int eighth_note_ms;
int sixteenth_note_ms;
int half_note_ms;
int whole_note_ms;

/*_____________________________________________________________________________

Prototypes
_____________________________________________________________________________*/

UINT SerialThreadFunction(LPVOID TFParam);                  //serial thread function
void SetTempo(int beats_per_minute);                        //function to set the current tempo
note_length_t FindNoteLength(int note_time_ms);             //function to determine a note length

/*_____________________________________________________________________________

class CMainWin:     Main Window of Application
_____________________________________________________________________________*/

CMainWin::CMainWin(LPCSTR ClassName) {

        //create a window with defined class style and no parent
        Create(     ClassName,
                    "Sheet Music Generator",
                    WS_OVERLAPPEDWINDOW|WS_MAXIMIZE ,
                    rectDefault,NULL,
                     "MainMenu");

        //DC for main window
        CClientDC DC(this);

        //get screen dimensions
        screen_width = GetSystemMetrics(SM_CXSCREEN);
        screen_height = GetSystemMetrics(SM_CYSCREEN);

        //create a memory DC compatible with this window
        m_memDC.CreateCompatibleDC(&DC);

        //create and select a bitmap to store the screen
        m_bmp.CreateCompatibleBitmap(&DC, screen_width, screen_height);
        m_memDC.SelectObject(&m_bmp);

        //create a standard brush background and use in virtaul window
        m_bkbrush.CreateStockObject(WHITE_BRUSH);
        m_memDC.SelectObject(&m_bkbrush);

        //paint background of virtual window
        m_memDC.PatBlt(0, 0, screen_width, screen_height, PATCOPY);

        //load menu accelerators
        if(!LoadAccelTable("MainMenu"))
```

```
                    //alert user upon error
                    MessageBox("Cannot Load Accelerators", "Error");

        //create pen and brush to draw score with
        m_ScorePen.CreateStockObject(BLACK_PEN);
        m_ScoreBrush.CreateStockObject(WHITE_BRUSH);


        //initialize serial stuff
        serial_thread = NULL;
        serial_thread_running=FALSE;
        serial_thread_stop_flag=FALSE;


}//constructor CMainWin

BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)

        //keyboard
        ON_WM_CHAR ()

        //window
        ON_WM_DESTROY()
        ON_WM_PAINT ()

        //menu
        ON_COMMAND(IDM_FILE_NEW, OnFileNew)
        ON_COMMAND(IDM_FILE_SAVE, OnFileSave)
        ON_COMMAND(IDM_FILE_OPEN, OnFileOpen)
        ON_COMMAND(IDM_FILE_CLOSE, OnFileClose)
        ON_COMMAND(IDM_FILE_EXIT, OnFileExit)

        ON_COMMAND(IDM_SCORE_KEY, OnScoreKey)
        ON_COMMAND(IDM_SCORE_STOP, OnScoreStop)

        ON_COMMAND(IDM_HELP_ABOUT, OnHelpAbout)

END_MESSAGE_MAP()

/*--------------------------------------------------------------------------------------
Menu Implementation
--------------------------------------------------------------------------------------*/
afx_msg void CMainWin::OnHelpAbout() {

        //create dialog box
        CAboutDialog diagObject("AboutDialog", this);

        //show dialog modally
        diagObject.DoModal();

}//OnHelpAbout

afx_msg void CMainWin::OnFileExit() {

        int response;

        //prompt user to exit
        response = MessageBox("Quit the Program?", "Exit", MB_YESNO);

        //if user opted to exit
        if(response == IDYES)           {


                    //close the current score if applicable
                    OnFileClose();

                    //send windows a message to close
                    SendMessage(WM_CLOSE);
```

```
        }//if(response == IDYES)

}//OnFileExit

afx_msg void CMainWin::OnFileNew() {

        //close the current score if applicable
        OnFileClose();

        //draw a new score
        DrawBlankScore();

        //set up a new score object
        curr_score=new CScore;

}//OnFileNew

afx_msg void CMainWin::OnFileSave() {

        //attemp to save score, if user cancels, don't care
        curr_score->SaveScore();

}//OnFileSave

afx_msg void CMainWin::OnFileOpen() {

        //close the current score if applicable
        OnFileClose();

        //first draw background
        DrawBlankScore();

        //create a new score object
        curr_score = new CScore;

        //load the score from the file
        curr_score->LoadScore();

        curr_score->DrawScore(this);

}//OnFileOpen

afx_msg void CMainWin::OnFileClose() {

        //check for saving current score
        CheckSave();

        //clear screen
        ClearScoreScreen();

}//OnFileClose


afx_msg void CMainWin::OnScoreKey() {

        //check to see that we are not already capturing
        if (serial_thread_running) {

                MessageBox("Already capturing!!!", "Invalid Action", MB_OK|MB_ICONEXCLAMATION);
                return;

        }//if (serial_thread_running)

        //start a new score
        OnFileNew();

        //attempt to open a connection on COM1 at 9600 baud
        if (serial_conn.Open(1, 9600)) {

                //create a new thread to handle the serial interaction
```

```
                serial_thread=AfxBeginThread(  SerialThreadFunction,          //AFX_THREADPROC
                                               this                           //LPVOID Param
                                               THREAD_PRIORITY_NORMAL,        //int InitPriority =
THREAD_PRIORITY_NORMAL
                                                                              0,
                                        //UINT StackSize = 0
                                                                              0,
                                        //DWORD dwFlags = 0
                                                                              NULL);
                                //LPSECURITY_ATTRIBUTES Security = NULL

                //set flag to indicate thread exists
                serial_thread_running=TRUE;

                //clear the flag used for stopping the thread
                serial_thread_stop_flag=FALSE;

                //set tempo
                SetTempo(120);

        }//if (serial_conn.Open(1, 9600))

        else
                AfxMessageBox("Failed to open port!");

}//OnScoreKey

afx_msg void CMainWin::OnScoreStop() {

        //check to see if we are capturing
        if (!serial_thread_running) {

                MessageBox("No capture to stop!!!", "Invalid Action", MB_OK|MB_ICONEXCLAMATION);
                return;

        }//if (!serial_thread_running)

        //set the flag to thread the stop
        serial_thread_stop_flag=TRUE;

}//OnScoreStop



/*-------------------------------------------------------------------------------------
Window Implementation
-------------------------------------------------------------------------------------*/
afx_msg void CMainWin::OnPaint(void){

        //obtain device context for this window
        CPaintDC  DC(this);

        //copy the virtual window on to the window
        DC.BitBlt(0, 0, screen_width, screen_height, &m_memDC, 0, 0, SRCCOPY);

}//OnPaint

afx_msg void CMainWin::OnDestroy(void) {

        //destroy the current score if necessary
        if (curr_score)
                delete curr_score;

}//OnDestroy

void CMainWin::ClearScoreScreen(void){

        m_memDC.PatBlt(0, 0, screen_width, screen_height, PATCOPY);
        InvalidateRect(NULL);
```

```
}//CMainWin::ClearScoreScreen

/*-------------------------------------------------------------------------------------
File Mainpulation Routines
-------------------------------------------------------------------------------------*/
void CMainWin::CheckSave(void){

	//if there is a score currently open
	if (curr_score!=NULL) {

		//if this score has been edited
		if (curr_score->IsEdited()) {

			//prompt user that score should be saved
			if(MessageBox("Save current score?", "Exit", MB_YESNO)==IDYES)
				OnFileSave();

		}//if (curr_score->IsEdited())

		//destroy the object that held the current score
		DestroyCurrentScore();

	}//if (curr_score!=NULL)

}//CMainWin::CheckSave

void CMainWin::DestroyCurrentScore(void){

	//destroy the current score and free all memory
	delete curr_score;

	//set reference to null
	curr_score=NULL;

}//CMainWin::DestroyCurrentScore


/*-------------------------------------------------------------------------------------
Drawing Routines
-------------------------------------------------------------------------------------*/
void CMainWin::DrawBlankScore(void){

	int screen_bottom=screen_height-20*BOTTOM_BORDER;
	int screen_right=screen_width-2*RIGHT_BORDER;
	int screen_left = LEFT_BORDER;
	int screen_top=TOP_BORDER;
	int cur_y = TOP_BORDER;
	int num_lines=0;

	//select drawing tools
	m_memDC.SelectObject(&m_ScorePen);
	m_memDC.SelectObject(&m_ScoreBrush);

	//draw outline
	m_memDC.MoveTo(screen_left, screen_top);
	m_memDC.LineTo(screen_left, screen_bottom);
	m_memDC.LineTo(screen_right, screen_bottom);
	m_memDC.LineTo(screen_right, screen_top);
	m_memDC.LineTo(screen_left, screen_top);

	//move down to first staff start
	cur_y=SCORE_START_Y;

	//draw staffs
	while(cur_y+STAFF_HEIGHT+BETWEEN_STAFFS<screen_bottom){

		//draw 4 lines
		for(num_lines=0;num_lines<4;num_lines++) {

			//move to beginning of line
```

```
                    m_memDC.MoveTo(screen_left+INNER_BORDER, cur_y);

                    //draw line
                    m_memDC.LineTo(screen_right-INNER_BORDER, cur_y);

                    //move to next line
                    cur_y+=BETWEEN_LINES;

                }//for(num_lines=0;num_lines<4;num_lines++)

                //draw the last line
                m_memDC.MoveTo(screen_left+INNER_BORDER, cur_y);
                m_memDC.LineTo(screen_right-INNER_BORDER, cur_y);

                //draw the clef
                DrawClef(cur_y);

                //move to next staff
                cur_y+=BETWEEN_STAFFS;

                //draw line

        }//while(cur_y+STAFF_HEIGHT+BETWEEN_STAFFS<screen_bottom)

        //cause window to be repainted
        InvalidateRect(NULL);

}//DrawBlankScore


void CMainWin::DrawClef(int y_pos){

        RECT r;
        CBrush HollowBrush;
        CBrush* oldBrush;

        //create and select a brush that won't fill shapes in
        HollowBrush.CreateStockObject(HOLLOW_BRUSH);
        oldBrush=m_memDC.SelectObject(&HollowBrush);

        //move to bottom of clef and draw vertical line
        m_memDC.MoveTo(CLEF_X, y_pos+CLEF_BOTTOM_OVERHANG);
        m_memDC.LineTo(CLEF_X, y_pos-STAFF_HEIGHT -CLEF_TOP_OVERHANG);

        //set up objects for arc
        r.top=y_pos-STAFF_HEIGHT -CLEF_TOP_OVERHANG;
        r.bottom=y_pos-2*BETWEEN_LINES-1;
        r.left=CLEF_X-CLEF_TOP_ARC_WIDTH;
        r.right=CLEF_X+CLEF_TOP_ARC_WIDTH;

        //draw top arc;
        m_memDC.Arc(r.left, r.top,r.right,r.bottom,CLEF_X,r.bottom,CLEF_X,r.top);

        //move bounding rectangle down for lower arc,widen
        r.top=y_pos-2*BETWEEN_LINES;
        r.bottom=y_pos;
        r.left-=3;
        r.right+=3;

        //draw next arc
        m_memDC.Arc(r.left, r.top,r.right,r.bottom,CLEF_X,r.top,r.right,(r.top+BETWEEN_LINES));

        //adjust rectangle for inner curve
        r.bottom-=2;
        r.left+=5;
        r.top+=4;

        //draw inner arc
        m_memDC.Arc(r.left, r.top,r.right,r.bottom,r.right,(r.top+BETWEEN_LINES-4), r.left, (r.top+BETWEEN_LINES-4));
```

```
        //adjust rectangle again for inner inner arc
        r.bottom-=2;

        //draw inner inner arc
        m_memDC.Arc(r.left, r.top,r.right,r.bottom,r.left,(r.top+BETWEEN_LINES-4), CLEF_X, r.bottom);

        //reselect old brush
        m_memDC.SelectObject(oldBrush);

}//DrawBlankScore

/*_____________________________________________________________________________

class CAboutDialog: About Box
_____________________________________________________________________________*/

//all implementation is default!


/*_____________________________________________________________________________

class CMainApp:  Main and Only Application Object
_____________________________________________________________________________*/

BOOL CMainApp::InitInstance() {

        CBrush bkbrush;

        //create a standard background brush
        bkbrush.CreateStockObject(WHITE_BRUSH);

        //register window style class
        LPCSTR cname = AfxRegisterWndClass(0,
          LoadStandardCursor(IDC_ARROW),
          bkbrush,
          LoadIcon("MainIcon"));


        //create a main window object, store pointer
        m_pMainWnd = new CMainWin(cname);

        //show the window object
        m_pMainWnd->ShowWindow(m_nCmdShow|SW_SHOWMAXIMIZED);

        //update the window object
        m_pMainWnd->UpdateWindow();

        //return succesfullly
        return TRUE;

}//CMainApp::InitInstance

/*_____________________________________________________________________________

class CNote:  Note Object
_____________________________________________________________________________*/

CNote::CNote() {

        //assign default values for length and pitch
        note_length = quarter_note;
        note_num=60;

        //next note not set until list created
        next_note=NULL;

}//CNote::CNote

CNote::CNote(int num,note_length_t length) {
```

```
		//initialize pitch and length to given values
		note_length = length;
		note_num=num;

		//next note not set until list created
		next_note=NULL;

}//CNote::CNote(int num,int length)

void CNote::SetNext(CNote* next_n){

		next_note=next_n;

}//CNote::SetNext

CNote* CNote::GetNext(void){

		return next_note;

}//CNote::SetNext



void CNote::DrawNote(int cur_x, int staff_base,CDC* memDC) {

		CBrush noteBrush;
		CBrush* oldBrush;
		CPen stemPen;
		CPen* oldPen;
		int cur_y=staff_base;
		int note_y_middle;
		RECT r;

		switch(note_num){

		case 60:
			cur_y+=BETWEEN_LINES;

			//draw line
			memDC->MoveTo(cur_x-C_LINE_OVERHANG,cur_y);
			memDC->LineTo(cur_x+NOTE_WIDTH+C_LINE_OVERHANG,cur_y);

			break;
		case 61:

			cur_y+=BETWEEN_LINES;

			//draw sharp sign and move over
			DrawSharp(cur_x, cur_y,memDC);
			cur_x+=(int)(BETWEEN_NOTES/4);

			//draw line
			memDC->MoveTo(cur_x-C_LINE_OVERHANG,cur_y);
			memDC->LineTo(cur_x+NOTE_WIDTH+C_LINE_OVERHANG,cur_y);

			break;
		case 62:
			cur_y+=(int)(BETWEEN_LINES/2);
			break;
		case 63:

			cur_y+=(int)(BETWEEN_LINES/2);

			//draw sharp sign and move over
			DrawSharp(cur_x, cur_y,memDC);
			cur_x+=(int)(BETWEEN_NOTES/4);

			break;
		case 64:
			break;
```

```
case 65:
        cur_y-=(int)(BETWEEN_LINES/2);
        break;
case 66:

        cur_y-=(int)(BETWEEN_LINES/2);

        //draw sharp sign and move over
        DrawSharp(cur_x, cur_y,memDC);
        cur_x+=(int)(BETWEEN_NOTES/4);

        break;
case 67:
        cur_y-=BETWEEN_LINES;
        break;
case 68:

        cur_y-=BETWEEN_LINES;

        //draw sharp sign and move over
        DrawSharp(cur_x, cur_y,memDC);
        cur_x+=(int)(BETWEEN_NOTES/4);

        break;
case 69:
        cur_y-=(int)(BETWEEN_LINES*3/2);
        break;
case 70:

        cur_y-=(int)(BETWEEN_LINES*3/2);

        //draw sharp sign and move over
        DrawSharp(cur_x, cur_y,memDC);
        cur_x+=(int)(BETWEEN_NOTES/4);

        break;
case 71:
        cur_y-=2*BETWEEN_LINES;
        break;
case 72:
        cur_y-=BETWEEN_LINES*5/2;
        break;
case 73:
        cur_y-=BETWEEN_LINES*5/2;

        //draw sharp sign and move over
        DrawSharp(cur_x, cur_y,memDC);
        cur_x+=(int)(BETWEEN_NOTES/4);

        break;
case 74:
        cur_y-=BETWEEN_LINES*3;
        break;
case 75:
        cur_y-=BETWEEN_LINES*3;

        //draw sharp sign and move over
        DrawSharp(cur_x, cur_y,memDC);
        cur_x+=(int)(BETWEEN_NOTES/4);

        break;

case 76:
        cur_y-=BETWEEN_LINES*7/2;

        break;

case 77:
        cur_y-=BETWEEN_LINES*4;
```

```
		break;

case 78:

		cur_y-=BETWEEN_LINES*4;

		//draw sharp sign and move over
		DrawSharp(cur_x, cur_y,memDC);
		cur_x+=(int)(BETWEEN_NOTES/4);

		break;

}//switch(note_num)

//make pen thicker for tail
stemPen.CreatePen(PS_SOLID, 2, RGB(0,0,0));

//set new pen, save old pen
oldPen=memDC->SelectObject(&stemPen);

//if note is to be filled in, create a black brush
if (note_length<=quarter_note)
		noteBrush.CreateStockObject(BLACK_BRUSH);
//otherwise create a hollow brush
else
		noteBrush.CreateSt ockObject(HOLLOW_BRUSH);

//save the old brush
oldBrush=memDC->SelectObject(&noteBrush);

//calculate note middle from staff bottom and note num
note_y_middle = cur_y;

//set up rectangle for note body
r.top=(int)note_y_middle-NOTE_HEIGHT/2;
r.bottom=(int)note_y_middle+NOTE_HEIGHT/2;
r.left=cur_x;
r.right=r.left+NOTE_WIDTH;

//draw oval for note body
memDC->Ellipse(&r);

//if note is not a whole note, it needs a stem
if (note_length<whole_note){

		//move to stem starting point
		memDC->MoveTo(r.right, note_y_middle);

		//draw stem up for lower notes
		if (note_num<=70) {

				memDC->LineTo(r.right, (int)(note_y_middle-NOTE_STEM));

				//if note is less than a quarter note, it needs a tail
				if (note_length<quarter_note)
						DrawTail(r.right,(int)(note_y_middle-NOTE_STEM-1),memDC);

				//if note is less than an eighth note, it needs a second tail
				if (note_length<eighth_note)
						DrawTail(r.right,(int)(note_y_middle-NOTE_STEM+TAIL_SPACING),memDC);

		}//if (note_num<=70)

		//draw stem down for higher notes
		else {

				//move to stem starting point
				memDC->MoveTo(r.left, note_y_middle);

				memDC->LineTo(r.left, (int)(note_y_middle+NOTE_STEM));
```

```
                                //if note is less than a quarter note, it needs a tail
                                if (note_length<quarter_note)
                                        DrawUpsideDownTail(r.left,(int)(note_y_middle+NOTE_STEM),memDC);

                                //if note is less than an eighth note, it needs a second tail
                                if (note_length<eighth_note)
                                        DrawUpsideDownTail(r.left,(int)(note_y_middle+NOTE_STEM-
TAIL_SPACING),memDC);

                        }//ekse


        }//if (note_length<whole_note)

        //set pen back to what it was
        memDC->SelectObject(oldPen);

        //reselect the old brush before returning
        memDC->SelectObject(oldBrush);

}//CNote::DrawNote

void CNote::DrawTail(int x_start, int y_start, CDC* memDC) {

        CBrush tailBrush;
        CBrush* oldBrush;
        CPen tailPen;
        CPen* oldPen;

        RECT r;

        //make brush that won't fill background
        tailBrush.CreateStockObject(HOLLOW_BRUSH);

        //set new brush, save old brush
        oldBrush = memDC->SelectObject(&tailBrush);

        //make pen thicker for tail
        tailPen.CreatePen(PS_SOLID, 2, RGB(0,0,0));

        //set new pen, save old pen
        oldPen=memDC->SelectObject(&tailPen);

        //set up rectangle for first arc
        r.top = y_start;
        r.bottom=y_start+TAIL_HEIGHT;
        r.left=x_start;
        r.right = x_start+TAIL_WIDTH;

        //draw first arc
        memDC->Arc(r.left,r.top,r.right,r.bottom, r.right,(int)(r.top+TAIL_HEIGHT/2),r.left,r.top);

        //adjust rectangle for second arc
        r.left=r.right;
        r.right+=TAIL_WIDTH;

        //draw second arc
        memDC->Arc(r.left,r.top,r.right,r.bottom, r.left,(int)(r.top+TAIL_HEIGHT/2),(int)(r.left+TAIL_WIDTH/2),r.bottom);

        //set brush back to what it was
        memDC->SelectObject(oldBrush);

        //set pen back to what it was
        memDC->SelectObject(oldPen);

}//CNote::DrawTail

void CNote::DrawUpsideDownTail(int x_start, int y_start, CDC* memDC) {
```

```
		CBrush tailBrush;
		CBrush* oldBrush;
		CPen tailPen;
		CPen* oldPen;

		RECT r;

		//make brush that won't fill background
		tailBrush.CreateStockObject(HOLLOW_BRUSH);

		//set new brush, save old brush
		oldBrush = memDC->SelectObject(&tailBrush);

		//make pen thicker for tail
		tailPen.CreatePen(PS_SOLID, 2, RGB(0,0,0));

		//set new pen, save old pen
		oldPen=memDC->SelectObject(&tailPen);

		//set up rectangle for first arc
		r.bottom = y_start;
		r.top=y_start-TAIL_HEIGHT;
		r.left=x_start;
		r.right = x_start+TAIL_WIDTH;

		//draw first arc
		memDC->Arc(r.left,r.top,r.right,r.bottom, r.left,r.bottom, r.right,(int)(r.top+TAIL_HEIGHT/2));

		//adjust rectangle for second arc
		r.left=r.right;
		r.right+=TAIL_WIDTH;

		//draw second arc
		memDC->Arc(r.left,r.top,r.right,r.bottom, (int)(r.left+TAIL_WIDTH/2),r.top,r.left,(int)(r.top+TAIL_HEIGHT/2));

		//set brush back to what it was
		memDC->SelectObject(oldBrush);

		//set pen back to what it was
		memDC->SelectObject(oldPen);

}//CNote::DrawUpsideDownTail

void CNote::DrawSharp(int start_x, int start_y, CDC* memDC){

		CPen sharpPen;
		CPen* oldPen;

		RECT r;

		//make pen thicker for tail
		sharpPen.CreatePen(PS_SOLID, 2, RGB(0,0,0));

		//set new pen, save old pen
		oldPen=memDC->SelectObject(&sharpPen);

		//set up base rectagle
		r.bottom=(int)(start_y+SHARP_HEIGHT/2);
		r.top=(int)(start_y-SHARP_HEIGHT/2);
		r.left=start_x;
		r.right=r.left+SHARP_WIDTH;

		//draw vertical lines
		memDC->MoveTo(r.left,r.bottom);
		memDC->LineTo(r.left+SHARP_SKEW,r.top);
		memDC->MoveTo(r.left+(int)(SHARP_WIDTH/2), r.bottom);
		memDC->LineTo(r.left+SHARP_WIDTH/2+SHARP_SKEW,r.top);

		//draw horizontal lines
```

```
		memDC->MoveTo(r.left-3,start_y+SHARP_OFF_LINE);
		memDC->LineTo(r.left-3+SHARP_WIDTH, start_y+SHARP_OFF_LINE);
		memDC->MoveTo(r.left-3,start_y-SHARP_OFF_LINE);
		memDC->LineTo(r.left-3+SHARP_WIDTH, start_y-SHARP_OFF_LINE);

		//set pen back to what it was
		memDC->SelectObject(oldPen);

}//CNote::DrawSharp

int CNote::GetNoteNum(){

		return note_num;

}//CNote::GetNoteNum

note_length_t CNote::GetNoteLength(){

		return note_length;

}//CNote::GetNoteLength

 /*_____________________________________________________________________________________________

class CScore:  Score Object
_____________________________________________________________________________________________*/

CScore::CScore() {

		//a new score has no notes yet
		note_list=NULL;
		last_note=NULL;

		//a new score has yet to be edited
		edited=false;

}//CScore::CScore

CScore::~CScore() {

		CNote* curr_note=note_list;
		CNote* next_note;

		//destroy all notes
		while(curr_note!=NULL){

			next_note = curr_note->GetNext();
			delete curr_note;
			curr_note=next_note;

		}//while(curr_note!=NULL)

}//CScore::~CScore

void CScore::AddNote (int num,note_length_t len){

		//create a new note
		CNote* new_note = new CNote(num,len);

		//if the list is empty, simply put in list
		if (last_note==NULL){

			note_list = new_note;
			last_note=note_list;

		}//if (last_note==NULL)

		//otherwise, make it the last note ande make old last point to it
		else {
```

```
                last_note->SetNext(new_note);
                last_note=new_note;

        }//else

        //mark that the score was edited
        edited=true;

}//CScore::AddNote

void CScore::DrawScore(CMainWin* main_window){

        int cur_x=SCORE_START_X;
        int cur_y=SCORE_START_Y+STAFF_HEIGHT;

        CNote* curr_note=note_list;

        //while there are notes to draw
        while(curr_note!=NULL){

                //draw the current note
                curr_note->DrawNote(cur_x,cur_y, &(main_window->m_memDC));

                //if note still fits on this line
                if (cur_x<LINE_LENGTH-2*NOTE_WIDTH) {

                        //move over one note width plus one space
                        cur_x+=NOTE_WIDTH;
                        cur_x+=BETWEEN_NOTES;

                }//if (cur_x<LINE_LENGTH-2*NOTE_WIDTH)

                //otherwise move to next line
                else {

                        cur_x = SCORE_START_X;
                        cur_y+=STAFF_HEIGHT+BETWEEN_ST AFFS;

                }//else

                //move to next note
                curr_note = curr_note->GetNext();

        }//while

        //cause the window to be repainted
        main_window->InvalidateRect(NULL);

}//CScore::DrawScore

bool CScore::SaveScore(void){

        if (curr_score==NULL)
                return FALSE;

        CNote* curr_note=note_list;

        CFileDialog SaveDialog(         FALSE,                                  //BOOL bOpenFileDialog
                                        ".txt",                                 //LPCTSTR lpszDefExt =NULL
                                        FILE_DEFAULT_NAME,                      //LPCTSTR lpszFileName
                                        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,         //DWORD dwFlags
                                         "Text Files (*.txt))|*.txt|",          //LPCTSTR lpszFilter
                                        NULL);                                  //CWnd* pParentWnd


        bool saved_succesfully=FALSE;
        int result;


        //show file save common dialog
```

```
	result = SaveDialog.DoModal() ;

	//if user cancelled, return without saving
	if (result==IDCANCEL)
		return saved_succesfully;

	//if file was selected correctly
	else if (result== IDOK){

		//open a stream for the file name selected
		ofstream SaveFile(SaveDialog.GetFileName());

		//write first line
		SaveFile << "File Created By Sheet Music Generator: Version 1.0\n";

		//while there are notes to save
		while(curr_note!=NULL){

			//save the current note
			SaveFile << curr_note->GetNoteNum() << "," << curr_note->GetNoteLength() << "\n";

			//move to next note
			curr_note = curr_note->GetNext();

		}//while

		//mark the end of the file
		SaveFile << "end";

		//close the file
		SaveFile.close();

		//mark score as not edited
		curr_score->edited=FALSE;

		//save now successful
		saved_succesfully=true;

	}//else if (result== IDOK)

	return saved_succesfully;

}//CScore::SaveScore

void CScore::LoadScore(void){

	CNote* curr_note=note_list;

	CFileDialog OpenDialog(	TRUE,			//BOOL bOpenFileDialog
				".txt",			//LPCTSTR lpszDefExt = NULL
				 NULL,			//LPCTSTR lpszFileName
				OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,		//DWORD dwFlags
				 "Text Files (*.txt))|*.txt|",	//LPCTSTR lpszFilter
				NULL);			//CWnd* pParentWnd


	int result;
	note_length_t temp_note_len;
	int temp_note_num;
	char file_buffer [FILE_BUFFER_LEN];


	//show file open common dialog
	result = OpenDialog.DoModal() ;

	//if user cancelled, return without opening anything
	if (result==IDCANCEL)
		return;

	//if file was selected correctly
```

```
	else if (result== IDOK){

		//open a stream for the file name selected
		ifstream OpenFileStream(OpenDialog.GetFileName());

		//get the first line
		OpenFileStream.getline(file_buffer,FILE_BUFFER_LEN,'\n');

		//if first line incorrect, bad file
		if (strcmp(file_buffer, "File Created By Sheet Music Generator: Version 1.0")!=0)
			return;

		//while we haven't hit the end of the file
		while(!OpenFileStream.eof()){

			//get the first int
			OpenFileStream.getline(file_buffer,FILE_BUFFER_LEN,',');

			//if we haven't reached the end of the file
			if (strcmp(file_buffer, "end")!=0){

				temp_note_num = atoi(file_buffer);

				//get second int
				OpenFileStream.getline(file_buffer,FILE_BUFFER_LEN,'\n');
				temp_note_len = (note_length_t)atoi(file_buffer);

				//add the note
				curr_score->AddNote(temp_note_num, temp_note_len);

			}//if (strcmp(file_buffer, "end")!=0)

		}//while(!OpenFileStream.eof())

		//close the file
		OpenFileStream.close();

	}//else if (result== IDOK)

	//score has yet to be edited
	edited=false;

}//CScore::LoadScore

bool CScore::IsEdited(){

	return edited;

}//CScore::IsEdited




 /*_____________________________________________________________________________

Main Code Section
_____________________________________________________________________________*/

CMainApp MyApp;                                                    //create
an application object to start


UINT SerialThreadFunction(LPVOID TFParam){

	int nBytesRead=0;
	unsigned int curr_note_num=0;
	unsigned int curr_note_len=0;
	int curr_byte=0;
	int byte_within_note=0;
```

```
unsigned char* lpBuffer = new unsigned char[500];

CMainWin* window_obj;

//cast the parameter into a serial connection object
window_obj = (CMainWin*) TFParam;

//continuously read the serial input
while(1) {

	//read any data waiting on the COM port
	nBytesRead = window_obj->serial_conn.ReadData(lpBuffer, 500);

	//process each byte read
	for (curr_byte=0;curr_byte<nBytesRead;curr_byte++){

		//decide which part of note current byte represents
		switch (byte_within_note) {

			//first byte is note num
			case 0:

				//grab and cast note number
				curr_note_num=(unsigned int)lpBuffer[curr_byte];

				//if note is below bottom of range, move up
				if ((curr_note_num>=48)&&(curr_note_num<=59))
					curr_note_num+=12;
				else if (curr_note_num<48)
					curr_note_num+=24;
				else if ((curr_note_num>=79)&&(curr_note_num<=90))
					curr_note_num-=12;
				else if (curr_note_num>90)
					curr_note_num-=24;

				//move to next byte in note
				byte_within_note++;

				break;

			//second byte is note length high byte
			case 1:

				//grab and cast note length
				curr_note_len=(unsigned int)(lpBuffer[curr_byte]);


				//move to next byte in note
				byte_within_note++;

				break;

			//third byte is note length low byte
			case 2:

				//combine high and low bytes
				curr_note_len = (curr_note_len*256)+(unsigned int)(lpBuffer[curr_byte]);

				//grab and cast note length
				curr_note_len=FindNoteLength(curr_note_len);


				//move to next byte in note
				byte_within_note++;

				break;

			//third byte is seperator
			case 3:
```

```
                                        //use this time to add the note
                                        curr_score->AddNote(curr_note_num,(note_length_t)curr_note_len);

                                        //move to first byte of next note
                                        byte_within_note=0;

                                        //draw the changes
                                        curr_score->DrawScore(window_obj);

                                        break;

                                }//switch

                        }//for

                        //if we've been asked to stop
                        if (window_obj->serial_thread_stop_flag)
                                break;

                }//while

                //delete buffer used
                delete []lpBuffer;

                //indicate that connection is no longer running
                window_obj->serial_thread_running=FALSE;

                //close the serial connection
                window_obj->serial_conn.Close();

                return 0;

}//SerialThreadFunction

void SetTempo(int beats_per_minute) {

                float quarter_note_sec;
                float eighth_note_sec;
                float sixteenth_note_sec;
                float half_note_sec;
                float whole_note_sec;

                //use tempo to calculate quarter note value
                quarter_note_sec = 60/(float)beats_per_minute;

                //use quarter note value to calculate other notes' values
                eighth_note_sec = quarter_note_sec/2;
                sixteenth_note_sec = eighth_note_sec/2;
                half_note_sec=quarter_note_sec*2;
                whole_note_sec = half_note_sec * 2;

                //convert values into ms
                quarter_note_ms = (int)(quarter_note_sec*1000);
                eighth_note_ms = (int)(eighth_note_sec*1000);
                sixteenth_note_ms = (int)(sixteenth_note_sec*1000);
                half_note_ms = (int)(half_note_sec*1000);
                whole_note_ms = (int)(whole_note_sec*1000);

}//SetTempo

note_length_t FindNoteLength(int note_time_ms){

                int midway_sixteenth_eighth = (int)((eighth_note_ms-sixteenth_note_ms)/2+sixteenth_note_ms);
                int midway_eight_quarter = (int)((quarter_note_ms-eighth_note_ms)/2+eighth_note_ms);
                int midway_qaurter_half = (int)((half_note_ms-quarter_note_ms)/2+quarter_note_ms);
                int midway_half_whole = (int)((whole_note_ms-half_note_ms)/2+half_note_ms);

                if (note_time_ms<=midway_sixteenth_eighth)
                        return sixteenth_note;
                else if((note_time_ms>midway_sixteenth_eighth)&&(note_time_ms<=midway_eight_quarter))
```

```
		return eighth_note;
	else if((note_time_ms>midway_eight_quarter)&&(note_time_ms<=midway_qaurter_half))
		return quarter_note;
	else if((note_time_ms>midway_qaurter_half)&&(note_time_ms<=midway_half_whole))
		return half_note;
	else
		return whole_note;

}//FindNoteLength

// Serial.cpp

#include "stdafx.h"
#include "Serial.h"

CSerial::CSerial()
{

	memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
	memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );
	m_hIDComDev = NULL;
	m_bOpened = FALSE;

}

CSerial::~CSerial()
{

	Close();

}

BOOL CSerial::Open( int nPort, int nBaud )
{

	if( m_bOpened ) return( TRUE );

	char szPort[15];
	char szComParams[50];
	DCB dcb;

	wsprintf( szPort, "COM%d", nPort );
	m_hIDComDev = CreateFile( szPort, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL );
	if( m_hIDComDev == NULL ) return( FALSE );

	memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
	memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );

	COMMTIMEOUTS CommTimeOuts;
	CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
	CommTimeOuts.ReadT otalTimeoutMultiplier = 0;
	CommTimeOuts.ReadTotalTimeoutConstant = 0;
	CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
	CommTimeOuts.WriteTotalTimeoutConstant = 5000;
	SetCommTimeouts( m_hIDComDev, &CommTimeOuts );

	wsprintf( szComParams, "COM%d:%d,n,8,1", nPort, nBaud );

	m_OverlappedRead.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
	m_OverlappedWrite.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );

	dcb.DCBlength = sizeof( DCB );
	GetCommState( m_hIDComDev, &dcb );
	dcb.BaudRate = nBaud;
	dcb.ByteSize = 8;
	unsigned char ucSet;
	ucSet = (unsigned char) ( ( FC_RTSCTS & FC_DTRDSR ) != 0 );
	ucSet = (unsigned char) ( ( FC_RTSCTS & FC_RTSCTS ) != 0 );
	ucSet = (unsigned char) ( ( FC_RTSCTS & FC_XONXOFF ) != 0 );
```

```
	if( !SetCommState( m_hIDComDev, &dcb ) ||
		!SetupComm( m_hIDComDev, 10000, 10000 ) ||
		m_OverlappedRead.hEvent == NULL ||
		m_OverlappedWrite.hEvent == NULL ){
		DWORD dwError = GetLastError();
		if( m_OverlappedRead.hEvent != NULL ) CloseHandle( m_OverlappedRead.hEvent );
		if( m_OverlappedWrite.hEvent != NULL ) CloseHandle( m_OverlappedWrite.hEvent );
		CloseHandle( m_hIDComDev );
		return( FALSE );
		}

	m_bOpened = TRUE;

	return( m_bOpened );

}

BOOL CSerial::Close( void )
{

	if( !m_bOpened || m_hIDComDev == NULL ) return( TRUE );

	if( m_OverlappedRead.hEvent != NULL ) CloseHandle( m_OverlappedRead.hEvent );
	if( m_OverlappedWrite.hEvent != NULL ) CloseHandle( m_OverlappedWrite.hEvent );
	CloseHandle( m_hIDComDev );
	m_bOpened = FALSE;
	m_hIDComDev = NULL;

	return( TRUE );

}

BOOL CSerial::WriteCommByte( unsigned char ucByte )
{
	BOOL bWriteStat;
	DWORD dwBytesWritten;

	bWriteStat = WriteFile( m_hIDComDev, (LPSTR) &ucByte, 1, &dwBytesWritten, &m_OverlappedWrite );
	if( !bWriteStat && ( GetLastError() == ERROR_IO_PENDING ) ){
		if( WaitForSingleObject( m_OverlappedWrite.hEvent, 1000 ) ) dwBytesWritten = 0;
		else{
			GetOverlappedResult( m_hIDComDev, &m_OverlappedWrite, &dwBytesWritten, FALSE );
			m_OverlappedWrite.Offset += dwBytesWritten;
			}
		}

	return( TRUE );

}

int CSerial::SendData( const char *buffer, int size )
{

	if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

	DWORD dwBytesWritten = 0;
	int i;
	for( i=0; i<size; i++ ){
		WriteCommByte( buffer[i] );
		dwBytesWritten++;
		}

	return( (int) dwBytesWritten );

}

int CSerial::ReadDataWaiting( void )
{

	if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );
```

```
	DWORD dwErrorFlags;
	COMSTAT ComStat;

	ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );

	return( (int) ComStat.cbInQue );

}

int CSerial::ReadData( void *buffer, int limit )
{

	if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

	BOOL bReadStatus;
	DWORD dwBytesRead, dwErrorFlags;
	COMSTAT ComStat;

	ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );
	if( !ComStat.cbInQue ) return( 0 );

	dwBytesRead = (DWORD) ComStat.cbInQue;
	if( limit < (int) dwBytesRead ) dwBytesRead = (DWORD) limit;

	bReadStatus = ReadFile( m_hIDComDev, buffer, dwBytesRead, &dwBytesRead, &m_OverlappedRead );
	if( !bReadStatus ){
		if( GetLastError() == ERROR_IO_PENDING ){
			WaitForSingleObject( m_OverlappedRead.hEvent, 2000 );
			return( (int) dwBytesRead );
			}
		return( 0 );
		}

	return( (int) dwBytesRead );

}
```