MICK: A Design Environment for Musical Instruments

by

Samuel H. Thibault

Submitted to the Department of Electrical Engineering and Computer Science

In Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 2001

Author____

Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by____

Bakhtiar J. Mikhak
Thesis Supervisor

Accepted by____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

MICK: A Design Environment for Musical Instruments
by
Samuel H. Thibault

Submitted to the
Department of Electrical Engineering and Computer Science

May 21, 2001

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A growing body of educational research has shown that children learn most effectively when they are engaged in designing and constructing things that are personally meaningful to them. Consequently, the challenge facing many researchers and practitioners has been to design a diverse collection of construction kits that support learning about many powerful intellectual and artistic ideas. In this thesis we report on the design and the implementation of a toolkit, called MICK, geared towards rethinking music education. MICK is a musical instrument construction kit that enables novices, particularly children, to design and build their own musical instruments. The electronics components and software tools in MICK make it possible to rapidly prototype a wide variety of instruments and other devices. The process of constructing a musical instrument with MICK also provides learners with many authentic opportunities for exploring and reflecting on important mathematical, scientific, and engineering ideas.

Thesis Supervisor: Bakhtiar J. Mikhak
Title: Research Scientist, MIT Media Laboratory

Acknowledgements

Over the year I worked on this project I received help and guidance from a wide range of people. In starting at the beginning, I must certainly thank Andy Begel, who guided me to what must be the one of the very best places I could have done my research anywhere: The Lifelong Kindergarten Group at the MIT Media Lab. Once there I began working with my thesis advisor, Bakhtiar Mikhak. His tireless efforts in providing direction and resources to my project were extraordinary. His guidance in both technical and theoretical ideas was critical to my success. I can not praise his abilities enough.

The rest of the group also gave both help and advice when needed. Mitchel Resnick guided me with high-level ideas to help situate my project. Chris Lyon did an amazing job working on the Serial Interface. When I (often) needed technical help, I turned to the remarkable abilities of Brian Silverman, Robbie Berg, Casey Smith, Rahul Bhargava, and Daniel Kornhauser. I would also like to thank the graduate students in the Hyperinstruments group, especially Roberto Aimi and Gili Weinberg, for their feedback. Finally, I am grateful to Amir and Javaneh, who were enthusiastic participants in the workshop.

In addition to all the people who helped me directly at the lab, I would also like to thank my family and friends for all the support they have given me. Not only this year, while working on my thesis, but throughout my studies at MIT. Thanks to all of them I can finally wear the button on June 8th that says, “I did it! – MIT GRAD”

Contents

List of Figures

List of Tables

“Better learning will not come from finding better ways for the teacher to instruct, but from giving the learner better opportunities to construct.”
-Seymour Papert

1 Introduction

Support for music and theater arts education continues to decline. Many key decision-makers in educational systems worldwide bolster support for mathematics, science, and technology curricula at the expense of music and theater arts programs. However, there are many compelling reasons for protecting the arts education in our schools. First, the arts are an important form of human expression in and of themselves. Second, the arts have deep connections with and are complementary to the more abstract ways in which we as humans describe and make sense of the world around us. Third, the arts have been one of the major driving forces behind many technological advancements. Finally, children's deep interest in music and the arts provides an authentic context for introducing them to many important ideas in mathematics, science, engineering and design.

These observations present us with both a challenge and an opportunity. The challenge is to find good ways of preserving what is best about the arts education while addressing the growing need for cultivating a technological fluency in our children. On the other hand, we have the opportunity to use technology to give children the tools they need to create not only their own art but also their own tools for creating art. In fact, the results from a large body of research from the constructionist learning community[5,12,13] have shown that children learn most effectively when they are engaged in creating things that they care about. One of the principle components of the research methodology in this community has been to create and evaluate construction kits and support materials that provide children with multiple new paths for making sense of the world and expressing themselves.

Following in the constructionist tradition, the research presented here focuses on the design and implementation of a construction kit for making a wide variety of musical instruments. A key observation underlying this thesis is that appropriate uses of technology can provide children with learning experiences that would fundamentally challenge our assumptions and our stance towards music education.

A closer look at this construction kit would be helpful in getting a better sense of the types of learning opportunities it provides. Let us consider an activity in which children build their own musical instruments. In this environment we fill the learners toolkit not only with traditional materials like wood and string, but with a broad set of electronic sensors capable of detecting touch, light, temperature, distance, motion, sound, etc. Moreover, we provide a software tool that allows users to easily map the input from the sensors to musical output. This environment provides not only the traditional characteristics of a musical activity, but also helps the user learn design skills, gain technological fluency, engage in deeper social interaction, and connect to important ideas in music, science, and engineering.

While there are many similarities between constructing an instrument from basic materials and constructing an instrument that also includes technological tools, the most significant trade-off exists in moving from acoustical to digitally created sounds. For example, in constructing a xylophone from wood the designer will need to work very hard to insure that each key of the xylophone is the correct size for the pitch it is intended to produce. This physical requirement is similar for other basic instruments like a guitar, where the tightening of the strings is the crucial element. Once the builder moves to using a set of electronic sensors the physical details change significantly. For example, the designer might replace a xylophone key with a touch sensor. Now, the most important element is not the exact size of the key but how the key is struck. The output pitch can easily be set in software to create the correct musical output. In essence, the designer must look at the quantitative values that exist in playing the instrument (what value does the sensor read when I strike it?), not just the intuitive quantitative properties that exist in a traditional instrument (if I hit the sensor hard the note is loud). By considering this additional element, the user can explore a set of musical mappings that are not restricted by the basic materials available. Moreover, the builder can engage in thinking about properties that go beyond intuitive physical relationships and use them to augment traditional properties.

By using this advanced musical instrument construction kit, the user can explore not only representations for musical instruments but also methods for representing musical ideas and music compositions. We could even consider an example where the instrument can read through music autonomously, though in a different representation than sheet music. One such example might center on a car-like instrument that drives over colored pieces of tape. Each color could represent a different note to the car. As the car drives over that color it sustains the specific pitch for as long as it is driving over that color. Thus a composer of a piece of music for this musical car would write his melody in the colored tape. Beyond this one example it is possible to imagine many other musical representations that are very different from traditional musical notation. This realm of possibilities provides the potential for many rich learning opportunities that do not simply fit into any one traditional discipline.

In utilizing this a more advanced toolkit for building instruments, the learner will become involved in exploring important ideas in engineering and gain technological fluency. In the interaction with sensors alone, the user will need to begin thinking about scientific ideas. In using a light sensor, for example, the designer will need to explore the types of values that the sensor returns. After experimenting with the behavior of the sensor, they will need to decide what value is a good divider between values that mean “light” and values that mean “dark”. They might decide to divide the returned sensor values into several small ranges to provide more detail. All of these simple interactions are important in learning about engineering and design decisions. Also, these electronic elements are probably common in a lot of elements the learner sees outside of this activity. Hopefully they will start to think about the ways they interact with other objects and what types of sensors are involved in those interfaces. Through this continued cycle of design, construction, and reflection, learners will build confidence in scientific and engineering techniques. They will develop intuitions about which uses are most effective for particular sensors, building materials, and functional mappings.

In consideration of using this tool in a classroom setting, it is worthwhile to briefly examine how activities with this toolkit may be incorporated into the available time in a curriculum. When it is necessary to fit the activities into a limited amount of time, it may be best to limit the breadth of the activity on any given day but work hard to maintain a continuity from day to day if the activity can be extended to multiple days. In a single short session, learners could delve into one specific aspect of the toolkit related to current work in a class. For example, an activity in a science classroom might examine how to build a sensor out of unusual materials and incorporate it into a musical instrument built with the toolkit. When there are less time restrictions, users could be permitted to explore the toolkit in whichever directions they choose. In creating their instruments, students would need to find solutions to specific implementation problems they encounter. In doing so, their desire to make their instrument perform well will drive their exploration of the underlying tools and scientific ideas that are needed.

The previous discussion has been the driving force for designing a new computational toolkit for music education. This document will detail the design rationale, implementation, evaluation, possible uses, and future directions of a Musical Instrument Construction Kit (MICK) through four chapters. The first of these chapters will describe a sample scenario that reflects the type of interaction we feel is possible based on the features of the tool and the activities we have observed. The next chapter provides a technical description of both the software and hardware components of MICK. In the third chapter, results from workshops and feedback sessions are described. We look at future directions for MICK in the last chapter.

2 Scenario

To help illustrate the type of interaction envisioned between a child and this new musical instrument construction kit we will develop a scenario in which a child uses the toolkit. In this example a child named Mick will explore building a musical instrument and then share the musical instrument he has created with his classmates at school.

Mick begins by opening his toolkit and seeing what types of objects are inside. He finds an assortment of sensors, an interface to connect the sensors to his computer, and software that enables him to program how his musical instruments will work. Mick decides that he will look at a few of the examples provided in a booklet that came with the kit before deciding exactly what he wants to build. The first example shows how to build a set of bongos. Each of the two drums is created from a light sensor. When the light sensor is covered the drum sounds. This simulates the action of striking a real drum. After building and playing the bongos for a short time, Mick changes the sounds the drums are producing to different drum sounds. He tries cymbals and other drums. Mick next looks at one other example – a small piano. Each key in the piano uses a touch sensor. When the sensor is pressed, a note is played for that key. While exploring this example, Mick decides to change the notes that the piano plays to a different scale. Then Mick gets another idea. He records his voice saying different words and replaces the notes with his recorded voice. Now he can form sentences by pressing the keys in the right order.

After playing with these two example instruments, Mick now feels ready to create his own instrument. He decides that he will try using a distance sensor and map the value of that distance sensor to a musical pitch. While experimenting with different ranges and different MIDI voices, Mick decides to only have the notes play when he blows into a wind velocity sensor. Now when he blows into the sensor, a note is played based on the value of the distance sensor. This correlation reminds Mick of a trombone so he decides to change the MIDI voice to a trombone sound. Finally, Mick decides to add volume control to his instrument. He maps the wind velocity sensor to the volume of the output so he can play loudly by blowing hard or play softly by blowing lightly. After a little bit of decoration, Mick feels that his instrument is complete.

Mick is proud of the instrument he has made and decides to take it to school and to show it to his classmates at school the next day during show and tell. During his presentation, Mick talks about his instrument and how he expanded it from one sensor to two. He talks about choosing the right sensor values for his instrument and how the values map to musical output. After he shows his instrument, the class talks about the interfaces that different instruments have. They also talk about how they interact with other objects besides musical instruments and how other types of interfaces are designed.

After school that day, Mick returns home inspired to build a new sensor for use in a musical instrument. By connecting two wires to a piece of fresh Play-Doh™ [11], Mick is able to measure a resistive value. As he sculpts the Play-Doh™, Mick is able to change the value. Much like his previous instrument, Mick maps this value to a musical pitch. However, due to the novel nature of the Play-Doh™ instrument, Mick would like to create a special way for notating songs he will play on the instrument. Mick decides to write the music much like a mathematical plotting. The value of the drawn function represents how "squeezed" the Play-Doh™ in the instrument appears at each point. This representation allows Mick to play through his song by executing the correct pattern of squeezes; Mick does not even need to know the details of standard musical notation in order to play the songs he writes. He has created his own independent representation.

The Scenario above presents many different types of interactions between Mick and the toolkit, yet the ideas for these sample interactions did not come from a vacuum. The workshops conducted as a part of this research, as well as other feedback and inspiration from other Media Lab groups, directly demonstrated or strongly anticipated all of these types of interactions. Reading through the remainder of this document, the reader may find it helpful to keep in mind the broad range of interactions that are possible, and how these interactions can positively affect learning.

3 Design Evolution and Implementation

The musical instrument construction kit (MICK) developed out of previous work done in the Epistemology and Learning Group and the Lifelong Kindergarten Group at the MIT Media Lab. Expanding on workshops[6] done with a small programmable device called a *Cricket*[2,8], initial prototypes utilized a desktop environment that contained tools for writing musical compositions and programming musical instruments that contained Cricket sensors and devices. After those beginning efforts, the project was revamped to utilize the PocketPC™ [14] as both a programming environment and controller for the musical instruments.

3.1 Background Work

The first ideas for the musical instrument construction kit evolved from earlier work done with the Cricket. The Cricket itself is a small computer only slightly larger than, and powered by, a 9-volt battery. A single Cricket is capable of powering two LEGO motors, monitoring two sensors, and controlling several additional devices. Crickets can also communicate with other Crickets or a computer interface using infrared light. A dialect of the Logo programming language is used to program the Crickets. The language includes procedure calls, simple control structures, and standard numeric operations. Logo also has functions for motors, sensors, timers, and playing tones.

There have been several expansions to the basic Cricket as well. The selection of sensors, once including only light and touch sensors, now provides sensors to detect temperature, reflectance, acceleration, and more. In addition, expansion devices are available which can be plugged into the Cricket's bus ports. One such device provides additional inputs for sensors, allowing four additional sensors per device beyond the original two.

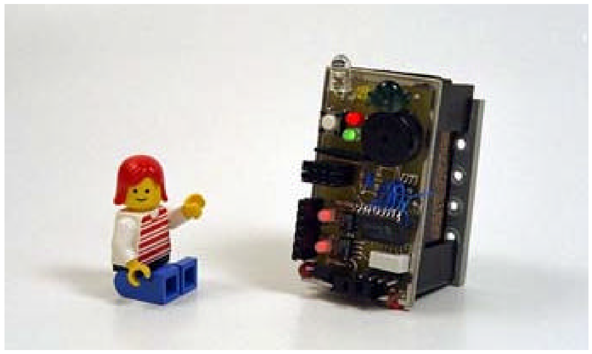


Figure 1: The Cricket

One recently created bus device that is especially useful for a Cricket based musical instrument is the Musical Instrument Digital Interface (MIDI) bus device[10]. With a set of primitive commands the Cricket can play musical notes using standard MIDI voices and channels, just like a musical synthesizer. MIDI commands are more complicated than other basic Logo commands and that makes MIDI programs much more difficult to write. More information about the Cricket may be found in Appendix A. More information on MIDI features may be found in Appendix E.

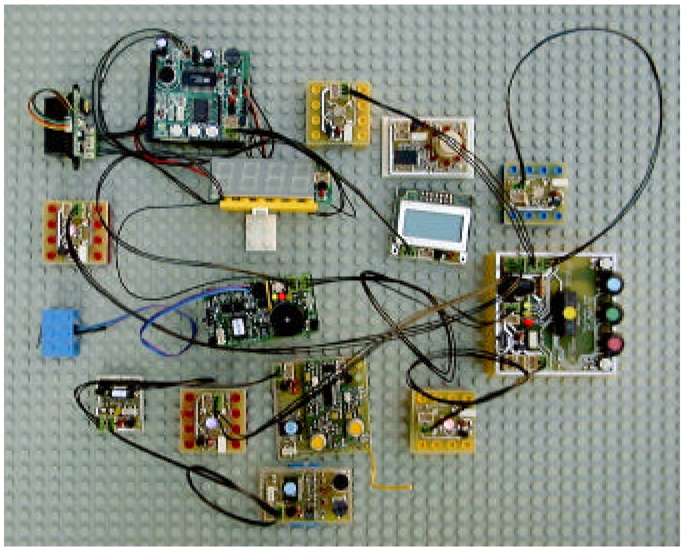


Figure 2: A Cricket with Bus Devices

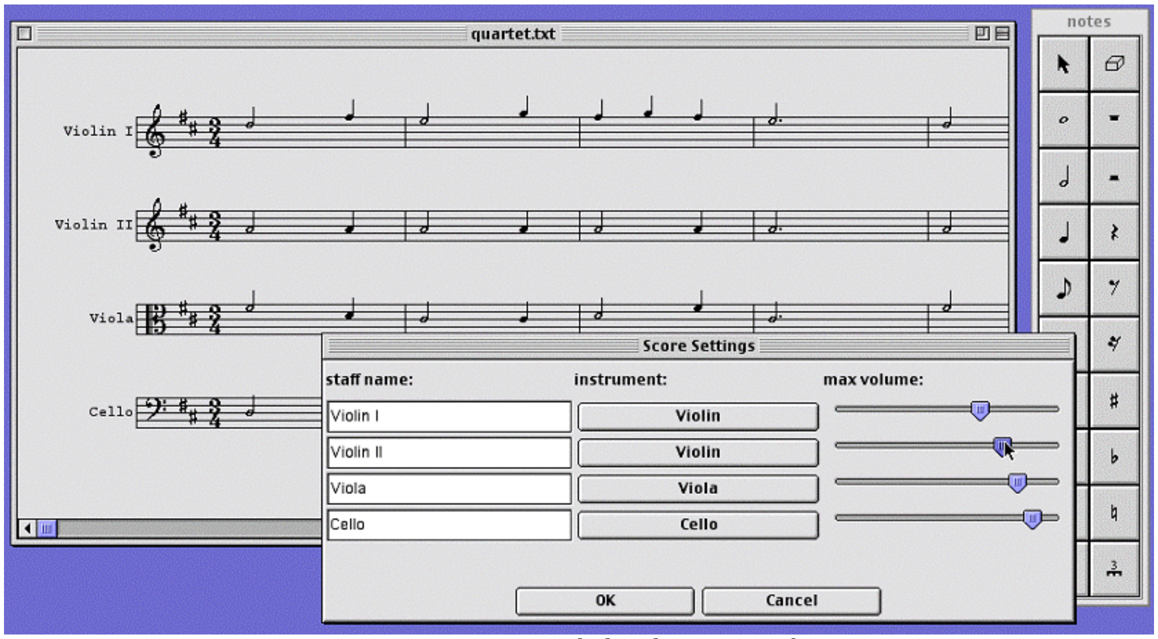
Some software tools have been written to aid in Logo programming. An application[3] written in the MicroWorlds™ [9] environment is currently the standard environment for programming the Cricket. It offers a command center interface for running code one instruction at a time as well as tools to load longer user programs onto the Cricket. LogoBlocks[1,7], another tool, provides a visual programming environment in which Logo commands and control structures are represented and manipulated as graphical blocks. A sequence of instructions is created be "snapping" the blocks together. However, neither of these tools currently has an easy method for programming MIDI commands.

This conflict between the common desire to build a musical instrument with the Cricket and the lack of an effective tool for creating a musical instrument became evident at a number of workshops with the Crickets. Although it was possible to build the instruments, the complexity needed to be relatively low and the time required to complete the instruments was often large.

3.2 Initial Prototype

The development of MICK began by focusing on improving the way in which a Cricket musical instrument was programmed. The new toolkit aimed at providing a simple interface for achieving musical effects. The first part of the project aimed at creating an environment for writing music compositions for playback on the Cricket. The second part of the project developed a graphical environment for programming a Cricket to behave like a musical instrument. Both of these elements were originally aimed at the desktop environment.

The melody editor provides a graphical interface for writing musical scores in standard notation. The user can select notes and markings from a palette and place them on a staff. The tool handles spacing the notes appropriately and positioning markings and notation in the appropriate place. The playback of the score can also be modified to use any set of MIDI voices. Once finished the melodies can be outputted as Logo code that can then be loaded onto a Cricket and be played back.



staff name:	instrument:	max volume:
Violin I	Violin	
Violin II	Violin	
Viola	Viola	
Cello	Cello	

Figure 3: Melody Editor Screenshot

The more significant half of the project was the instrument editor. The instrument editor allows the user to map sensor actions to musical output and other MIDI effects. For each sensor, the user can define a range for the effect to occur. When the sensor's value enters that range, the musical effect occurs. This effect could be playing a note or a chord, playing a melody, or performing some other Cricket action like turning on a motor. Though it was possible to quickly design an instrument with this interface, the Cricket suffered from a lack of processing power and a severe bottleneck in communication with the connected devices. Therefore, a new solution was developed in which the processing of the Cricket would be replaced with a PocketPC^{TM}. In addition a new interface would be designed to allow the PocketPC^{TM} to communicate with the same devices as the Cricket. These improvements were completed for the current version of MICK described below.

3.3 The MICK Environment

The full version of the project is called the Musical Instrument Construction Kit (MICK). It uses the PocketPC^{TM} rather than the desktop as the main working platform. To enable the PocketPC^{TM} to work with the Cricket's sensors, motors, and bus devices a special Serial Interface was designed. The initial instrument creation software created for the desktop was recreated and improved in a version for the PocketPC^{TM}. The melody-editing tool remained fairly untouched, though it was modified to download melodies to the PocketPC^{TM}.

In order to allow the PocketPC^{TM} to connect with Cricket sensors and bus devices, a new interface needed to be made with the PocketPC^{TM}'s native serial port. This interface would need to receive bytes from the PocketPC^{TM} that specify actions for the interface to perform, like checking a sensor's value, and then send bytes back with the specified result. There were three such actions that the interface needed to implement: sensor checks, midi commands, and bus commands.

The top two bits of each byte specify the action type; the remaining six bits contain information relevant to that command. For sensor checks, the low bits describe the sensor port to check. That port is checked and the value is returned in a byte. For a MIDI command the low bits specify how many MIDI instructions of one byte each, either two or three, will follow. Commands for bus devices use the lower bits to describe how many bytes will be sent to the bus device, and then how many bytes will be read back to the PocketPC^{TM}.

Command Type	Byte Appearance	Comments
Sensor Check	`00XXXPPP`	PPP is three bit id (0-7) of the sensor port to check. For example, '00000010' means check sensor port 2.
MIDI	`01XXXXXB`	B is number of bytes in the MIDI command: 0 = 3 byte command 1 = 2 byte command
Bus Device	`10RRRSSS`	RRR and SSS are 3 bit values (0-7). SSS is the number of bytes to send to the bus device. RRR is the number of byte to receive from the bus device.

Table 1: Serial Interface Byte Commands

Sensor ports on the Serial Interface provide ports identical to the ports on the Cricket. The interface provides room for eight sensors directly on the board, though additional sensors can be provided with bus devices. Sensor values are calculated using analog to digital conversion on a PIC^{TM} processor. For complete assembly code, block diagram, and PCB layout of the interface board see Appendix B.

One problem encountered in connecting the interface to standard Cricket bus devices was the communication protocol. Whereas the PocketPC^{TM} communicates with the Serial Interface using bytes (8 bits each), the Cricket communicates with bus devices using a 9 bit instruction, eight bits of data plus an additional command bit. After synchronizing with the bus device, the Cricket begins by sending a start bit, one high bit. Next follows 8 bits of data. The tenth bit of the signal is a command bit. This command bit is zero if it is the first bit sent to a bus device and one otherwise. The last bit sent is a low bit, the stop bit (see Figure 4).

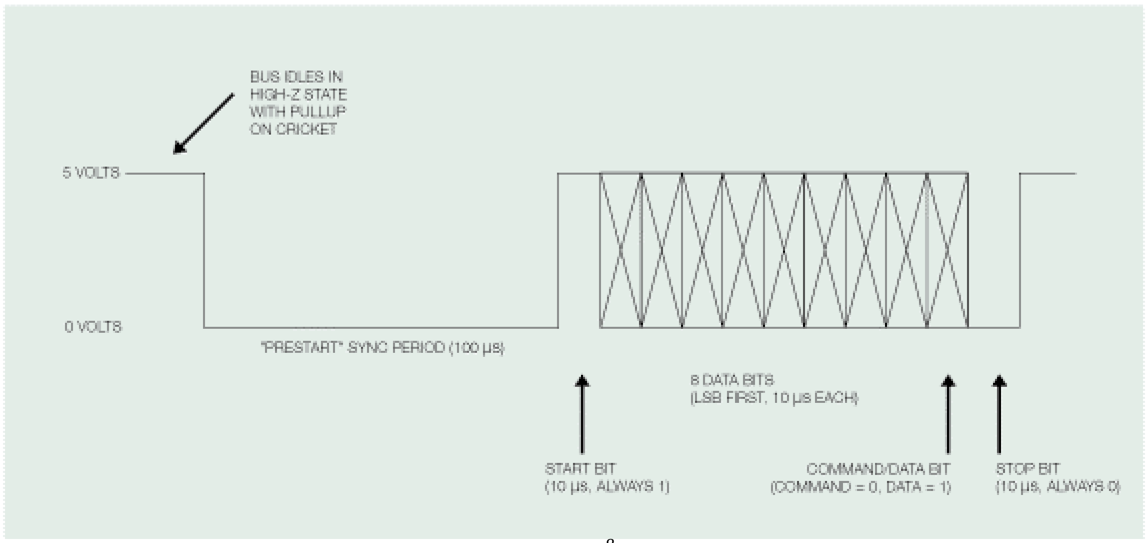

BUS IDLES IN HIGH-Z STATE WITH PULLUP ON CRICKET
5 VOLTS
0 VOLTS
"PRESTART" SYNC PERIOD (100 µs)
8 DATA BITS (LSB FIRST, 10 µs EACH)
START BIT (10 µs, ALWAYS 1)
COMMAND/DATA BIT (COMMAND = 0, DATA = 1)
STOP BIT (10 µs, ALWAYS 0)

Figure 4: The Bus Device Protocol[8] *(reprinted with permission)*

In order to account for the command bit, the Serial Interface makes the command bit low for only the first command sent to the bus devices from the Serial Interface. The eight bits of data are exactly the contents of the bytes sent from the PocketPC™ to the interface. The interface already knows how many bytes it will send from the original command byte (see Table 1 above).

The instrument editor on the PocketPC™ is very similar to the interface of the initial prototype for the desktop environment. Sensor values may still be mapped to musical output in the same way. The interface now provides many improvements over the original prototype. In addition to MIDI output, sounds recorded as wave files can also be outputted. Bus devices also provide an additional method of creating output whether through motors, displays, are other devices. One additional feature allows sensor values to be mapped through a mathematical function to output.

Because of the number of devices that could be attached to the Serial Interface, it is critical that the exact location of those devices be specified. MICK allows the user to describe where devices are located using simple dialog boxes. For sensors connected to the interface, naming the port is all that is required. For bus devices, the type and coloration of the bus device (red, blue, yellow, or white) is also needed.

The sensor ports on the Serial Interface to the PocketPC^{TM} replicate the ports existing on Crickets. Therefore sensor values can be treated exactly the same way as in the initial prototype. Sensors are divided into two basic categories: toggle and ranged. Toggle sensors only encompass touch sensors. For these sensors, the value returned to the Cricket is a value of true or false. Actions may be triggered for either of those two states. For example, we might want to say "when touch sensor #1 is pressed then do some action."

For ranged sensors, the value returned to the Cricket is an integer in the range from 0 to 255. For these sensors it is necessary for the user to specify what values will trigger an action by setting up a *range*. This range would have an upper and lower bound within the range 0 to 255. For example, we might want to say "when light sensor #2 is returning a value between 100 and 200 then do some action."

In the examples above, the "do some action" phrase has not yet been clarified. The user selects the desired actions from a set of options that are displayed, such as playing a note, playing a melody that was created in the melody editor, or causing some MIDI effect to occur. Once the type of action has been selected the user then supplies the details of that action. For example, in the case where the user would like a note (or set of notes to be played), a window appears where the user enters the notes to be played. If the user selected to play a melody, a box would appear where the filename for that melody should be entered.

In addition to MIDI output through a speaker connected to the Serial Interface, the PocketPC^{TM} includes an internal speaker that allows for additional media output. The most direct application of this capability is triggering wave file playback as output. This feature allows for many interesting instruments, such as a child creating an instrument that outputs sounds recorded from their own voice.

The ability to use bus devices for additional output provides a whole new scope for the type of projects created with MICK. Not only basic musical output may be performed, but control of motors and other devices. Instead of making merely a musical interface, a student could create an interface for controlling a car or robot. This functionality allows MICK to be used as a general-purpose tool in addition to just a musical tool.

One significant feature added in the PocketPC^{TM} that goes outside any feature in the initial desktop version is a tool for creating functional mappings. In a functional mapping, a sensor's value is passed through a mathematical function to return a new value. It is the returned value that may then be used for other effects. For example, a light sensor might be used with a functional mapping to control the instrument's volume. These mappings do not need to be linear. MICK provides four basic functions for mappings: linear, inverse, square, and square root.

The complete manual for MICK and all the possible actions it performs with examples is given in Appendix C.

4 Evaluation

MICK was demonstrated and used in several environments to get feedback on its functionality and usability. This feedback came from workshops conducted with middle school children ages eleven to thirteen, as well as through comments from other researchers in the MIT Media Lab. Particularly useful feedback was received from members of the Hyperinstruments Group directed by Tod Machover.

4.1 Workshop Results

The workshop consisted of three main phases. At the beginning, MICK was introduced to the students through some example instruments and a brief tutorial. Most of the time in the middle was dedicated to allowing the students to design and build an instrument from the tools and materials provided. At the end of the workshop each participant shared the instrument they had created.

The workshop began by introducing the students to two previously built examples. The first example was a simple piano. The piano consisted of five keys, with each key having an associated light sensor. The sensor was normally covered, but when a key was pressed the light was revealed to the sensor. The piano was set up to play one section of a standard scale with no special modifications. The second example was built to explore the realm of non-traditional instruments. It consisted of a raised rail with a moving car. An optical distance sensor was attached to the car facing downwards, with enough open space existing below the car to allow blocks to be stacked to different heights. As the car moved back and forth on the rail, the distance sensor would measure the distance to the stack of blocks directly below it and then map that value to a note. A tall stack of blocks would result in a high pitch, while a low stack of blocks would result in a low pitch. In addition, the instrument possessed a display that showed the value the sensor was reading. The display was useful in debugging the instrument and explaining the instruments operation to the students.

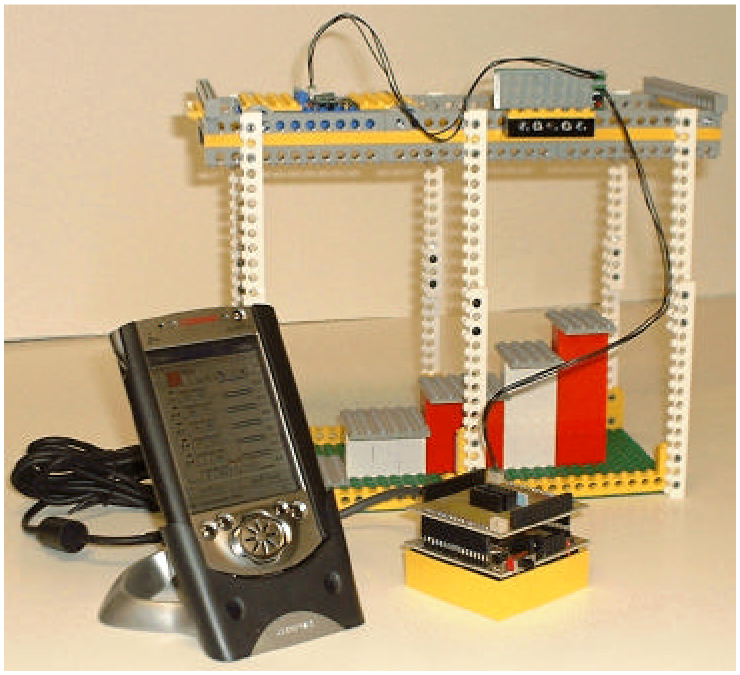


Figure 5: Sliding Car Instrument

After demonstrating these two instruments, the students were introduced to the software interface through a tutorial. The participants were shown how to start a new instrument and set the port location for the sensors they added. Next, the students went through the step-by-step process of setting up a light sensor with different ranges to play different notes. Following this introduction to MICK, the students began to design and build their own musical instruments. The creations of two of the students are described in detail below.

One of the students, named Andy, was interested in building an instrument that behaved like a guitar. He began by choosing the type of sensors he wanted to use. Andy decided touch sensors would work well for emulating the frets of the guitar as well as the strings of the guitar. Andy's next step was building the body of the guitar out of LEGO building blocks and embedding three touch sensors to represent strings, and two touch sensors in the neck of the guitar to use for frets. Now it was time to begin programming the guitar so the sensors would trigger sounds. The first step in programming the guitar was setting up one of the touch sensors representing a string. Since the default MIDI voice of the instrument sounded like a piano, Andy immediately changed the MIDI voice to sound like a guitar. After the first string was ready, Andy programmed the other two strings two sound at higher pitches than the first. Now Andy moved to the frets of the guitar. With a little help, Andy learned how to shift the pitch of other notes that were playing. Using this feature of the toolkit, he had the touch sensors on the fret modify the pitch of the notes played by the string sensors. This behavior was very true to the actual operation of a real guitar. After completing this first small guitar, Andy began building a more complete guitar with six strings and more frets.

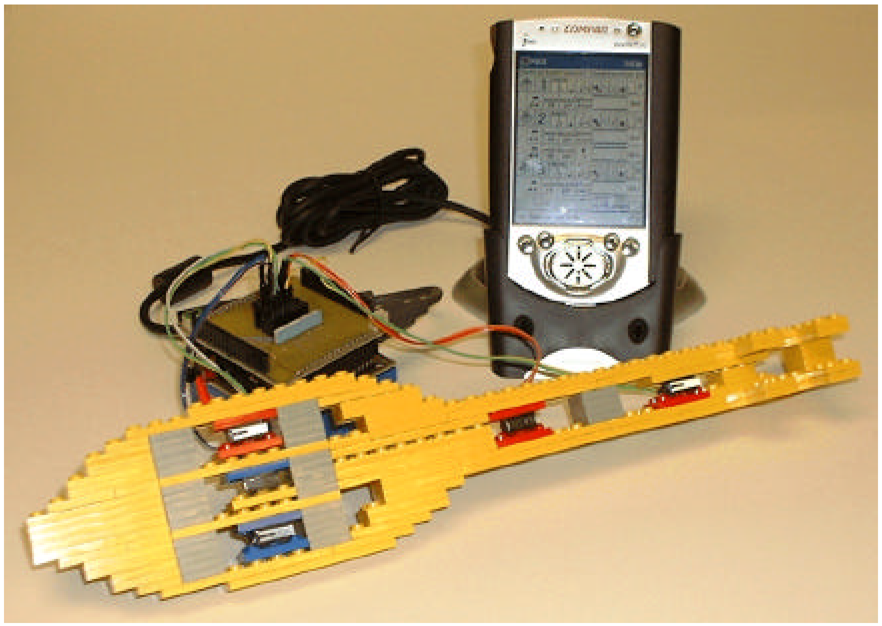


Figure 6: Guitar Instrument

Another participant at the workshop, named Jessica, built a very unusual instrument. She wanted to use a temperature sensor in her instrument. First she thought about what would be a good way to get different temperature readings. She quickly decided to use bowls of different temperature water. She filled three bowls with water: one warm, one medium, and one cold. She had seen the display used with the sliding car instrument and decided to use the same approach to help figure out what ranges to use in distinguishing the different temperatures of water. She measured each bowl of water with the sensor and picked a wide enough range to insure that she would identify the different bowls of water. Jessica first assigned a sound to the cold water. She wanted the instrument to play a high screeching pitch like someone would make when the feel really cold water. For the medium temperature she picked a chord that was in the middle; for the warm water she picked a very soothing chord. Once the instrument was playing the right sounds with one sensor she began experimenting with using a second temperature sensor. Eventually, she decided that just one sensor was best. Finally, Jessica completed her instrument by adding extra decorations with colored pipe cleaners.

Overall, the workshop went very well. All of the participants were very excited about the instruments they were creating and very happy with the results. None of the participants had particularly strong musical backgrounds, but they did not have problems using what they did know from just listening to music and seeing instruments played to get started on their own project. Also, the students felt they gained knowledge about music as well as about using electronic sensors.

Even in the limited time of this preliminary workshop, participants were able to complete a first version of their instruments and have many more ideas for other instruments. If given the possibility to continue interacting with this toolkit, these users would have opportunities to learn additional ideas in music, science, and engineering design. Their competency with basic tools could enable them to explore more sophisticated constructions and ideas. We suspect that over time users will reach fluency with the material in the toolkit. They will be able to talk competently about what they have created, in terms of musical and technical properties, and what their process was in doing so. Moreover, they will be able to consider other alternate ways of designing their instrument and evaluate those designs.

4.2 Additional Feedback

Beyond just the reactions from the students at the workshop, teachers, parents, and other researchers at the lab made comments about the musical instrument construction kit. While many of the comments were related to technical aspects of the system, other comments addressed ways of using the system in different activities.

Many of the technical comments addressed the ability of the system to affect fine detail and expression in performance. The chief observation was that simple MIDI was not capable of performing a high enough level of detail for use in genuine performance situations. While this fact is certainly true, the instruments we expect that students will design with this system would be more closely matched with the style of instruments a student would make from traditional materials. Neither of these categories of instruments would probably be seen on a concert hall stage. Nevertheless, it may be possible to modify the toolkit to bridge this existing gap between very sculpted and advanced technological instruments capable of true performances and the MIDI producing instruments made from the toolkit.

Another comment addressed the set of actions available to perform in the toolkit. Other forms of media output, like displaying video to the screen, were suggested. Also, the ability to change high-level properties of the instrument with a sensor mapping was also proposed. These might include switching MIDI voices or changing the behavior of the instrument based on some sensor mapping.

Musical representations were also addressed in some of the comments. In the programming environment, the user is forced into dealing with the standard musical notation that includes staffs and clefs. Several people thought that providing other ways of indicating the notes to be played could be effective, like using drawings or colored mappings.

5 Future Directions and Conclusions

Currently we have a stable software environment in MICK for writing music and creating musical instruments, as well as a tested set of hardware components for enabling the construction of a wide variety of instruments. Through both our personal evaluation process and outside feedback we have identified a number of ways in which to improve MICK.

Hardware

Expanding the variety of sensors available will quickly enhance the types of instruments that users can create. For example, a simple wind sensor would allow the construction of brass and woodwind style instruments that successfully replicate the feel of their traditional counterparts. We would like to extend the Serial Interface with two additional boards (stackable layers). The first is an improved version of our MIDI controller board. The second is a digital sound sampling and playback board.

Software

We would also like to make the software more powerful through steps in three important directions. First, we would like to provide ways of representing ideas and information in new and novel ways. Second, we would like to allow users to interact with the large body of music that already exists. Third, we would like to provide a scripting tool that enables more sophisticated instruments and a broader set of activities.

While MICK's interface allows users to program their instruments using standard musical notation and mathematical formulas, it would be nice to provide tools that are more imaginative and variable. For example, instead of modifying a standard mathematical function, the designer could simply draw a function, standard or unusual, which could then be used in their mathematical mappings. Similarly, we would like abstract ways of representing musical pitches and expressions. Such a representation might make use of color mappings or constructions in three-dimensional spaces. Simply expanding the realm of possibilities could spark an entire set of new ideas in people using MICK, and thereby create an interesting new set of non-traditional instruments.

Allowing users to incorporate the large body of already existing musical repertoire is also important. To this end we would like to incorporate tools for importing MIDI files into the system. Users would then be able to play their instruments along with those files. This type of interaction will engage users in a much richer performance environment.

The last major improvement we suggest here has been thought about in great detail. A tool for scripting a series of device commands would provide a significant expansion to the current interface. Rather than only performing a single action, the designer could trigger a set of actions. For example, the instrument would be able to blink lights in some repeating sequence or perform a sequence of motor actions. This expansion to MICK would be especially useful for creating general-purpose tools like an interface for controlling a robot, driving a car, or playing a video game. The most intricate aspect of the scripting environment is providing a method of multithreading the commands so that the sensor actions can cause multiple sequences to interleave, as well as supporting delays and procedure calls. Conveniently, most of these characteristics can be maintained by storing the command sequences in separate lists for each sensor and expanding calls in a lazy (waiting until required for execution) fashion. When a sensor re-enters a range, the user will probably want to decide whether to append the sequence to the end of the list or clear the list and begin the sequence again. Other questions include the use of global and local variables to avoid race conditions and other problems. A further discussion of such a scripting language can be found in Appendix D.

Activities

This toolkit could provide a wide set of activities in both music and science classrooms. For example, a very rich activity in a science class, which relates to the musical instrument construction, could be to explore how everyday materials (such as Play-Doh^{TM}, dish soap, fruits and vegetables, etc.) can be used to create novel sensors. In turn, those sensors can be used to create very whimsical musical instruments. This could naturally lead to a discussion about appropriate representations for notating and playing music for such an instrument. Furthermore, this could lead to an interesting discussion on the history of musical instruments and musical notation in the music classroom. In this direction, an immediate future project is to develop detailed activity booklets and support materials. In addition to schools, MICK can also be introduced into a clubhouse or after-school setting to provide kids with a chance to explore their ideas and promote their social interaction by playing the instruments they have created in small ensembles.

Conclusions

The focus of the project thus far has been on creating a powerful construction kit that highlights the interplay between many important ideas in music, science, and engineering design. In this thesis we presented the design rationale and implementation of MICK. We also discussed our preliminary findings from a number of studies and discussions with children, schoolteachers, and professional musicians. While we have had a lot of encouraging results from these interactions, we believe that there remains a need for a careful study of what learning opportunities these tools afford, and what implications they have for all aspects of our educational system.

For updates on the development of this project go to http://llk.media.mit.edu/projects/MICK/.

References

1 Andy Begel. *Logoblocks: A graphical programming language for interacting with the world*. MIT Media Laboratory, 1996.

2 Cricket, see http://llk.media.mit.edu/projects/cricket/

3 Cricket Logo, http://llk.media.mit.edu/projects/ProgrammableBricks/Home.htm

4 F&B PIC^{TM} Programmer, see http://www.media.mit.edu/llk/projects/picdev/

5 Aaron Falbel. *Constructionism: Tools to Build (and Think) With.* LEGO DACTA, 1995.

6 Carol E. Foltz. *Learning Through Design of Programmable Musical Instruments.* MIT Media Laboratory, 1996.

7 Logo Blocks, see http://llk.media.mit.edu/projects/summaries/bbp.shtml

8 Fred Martin, Bakhtiar Mikhak, and Brian Silverman. *MetaCricket: A Designer's Kit for Making Computational Devices*. IBM System Journal, VOL 39, NOS 3&4, 2000.

9 MicroWorlds^{TM} Software, see http://www.microwolrds.com

10 MIDI Board, see http://www.media.mit.edu/~jrs/minimidi/

11 Music Shapers and Toys, see http://www.media.mit.edu/hyperins/projects.html

12 Seymour Papert. *Mindstorms*. Basic Books, Inc., New York, New York, USA, 1980.

13 Seymour Papert. *What's the Big Idea? Steps Toward a Pedagogy of Idea Power*. IBM Systems Journal, 2000.

14 PocketPC^{TM}, see http://www.compaq.com/pocketpc/

Appendix A: The Cricket

To make this document self contained, in this appendix we will provide some technical detail about the Crickets in so far as they influence the design of the Serial Interface for this project. For a more detailed discussion of the Cricket system, please refer to the MetaCricket Paper from which the contents of this appendix have been reproduced (with permision).

The Cricket. The Cricket is a tiny, programmable computer (about the size of a 9-volt battery) that can directly control motors and receive information from sensors. The Cricket evolved from earlier MIT "Programmable Brick" designs, which have led to the recently introduced LEGO™ Mindstorms™ Robotics Invention System™ with its RCX™ Brick.

The Cricket is based on a Microchip PIC™ microprocessor. Basic actuators like DC (direct-current) motors and lightbulbs plug into one of the Cricket's two motor outputs, and simple resistive sensors such as switches, photocells, and thermistors plug into the Cricket's two analog voltage-sensing inputs.

All Cricket devices have a built-in bidirectional infrared communications channel, which is used for Cricket-to-desktop communication (when downloading programs to a Cricket, or viewing sensor data) and Cricket-to-Cricket communication. The Cricket also includes a peripheral expansion port, or "bus port." The use of this port greatly expands the capability of the Cricket and is discussed in depth later in this paper.

Feature	Description
Program size	2048 bytes of compiled code Each user-level primitive function compiles to 1, 2, or 3 bytes
Procedures	Arbitrary number of numeric inputs allowed May provide numeric return value
Number system	16-bit integers Add, subtract, multiply, divide, remainder, and modulus operators Greater than, less than, equality operators And, or, not, and exclusive-or operators Random number generator
Data and variables	16 available global variables Local variables (limited by stack depth) One-dimensional arrays (2048 bytes total array data, Persistent through power cycling
Control structures	If-then; if-then-else Loops (repeat *n* times or infinite) Waituntil (*Boolean expression*)
Multitasking	One foreground thread plus one background daemon Daemon fires when provided Boolean expression makes False-to-true transition 15-bit background millisecond timer (4-millisecond ticks)
Communications	Integrated infrared (IR) program download protocol Low-level primitives for IR communication between Crickets Low-level primitives for peripheral bus communication
Hardware-specific	Motor power, direction Analog input Boolean input Piezo tones

Table 2: Cricket Logo Feature Overview

Of particular importance to this project is the set of bus device that have been created and may be used with MICK. A partial listing of these devices is given below.

** DC Motor Controller

** Servo Motor Controller

** Numeric and Alphanumeric Displays

** Tri-Color LED and Mater Controller

** IR Transceiver Boards

** RF Transceiver Boards

** Additional Resistive Sensor Ports

** Optical Distance Sensor

** Reflectance Sensor

** LEGO Rotation Sensor

** Voice Recorder and Playback Module

** Heart Rate Monitor

** Sonar Range Sensor

** Clap and Pitch Sensor

** Keypad

** Digital Compass

** 2-axis Accelerometer

Appendix B: Serial Interface

Hardware

Provided below is a schematic of the initial interface board. The interface is based on a PIC16F876 processor and uses a MAX233 to communicate over the serial cable with the PocketPC^{TM}. Assembly (PIC^{TM}) code describing its operation is also provided.

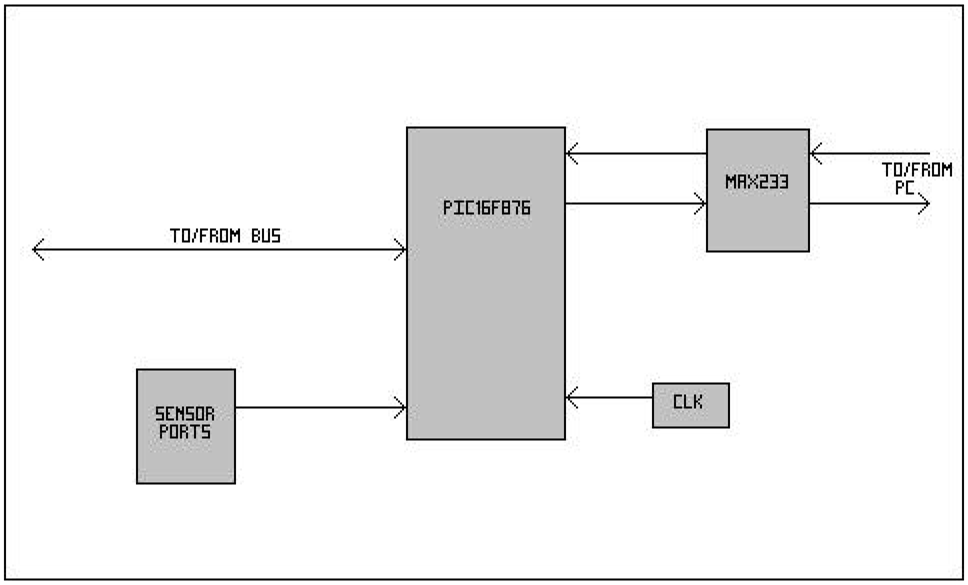
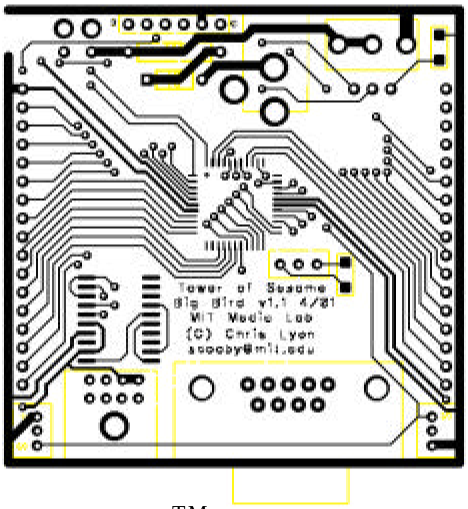
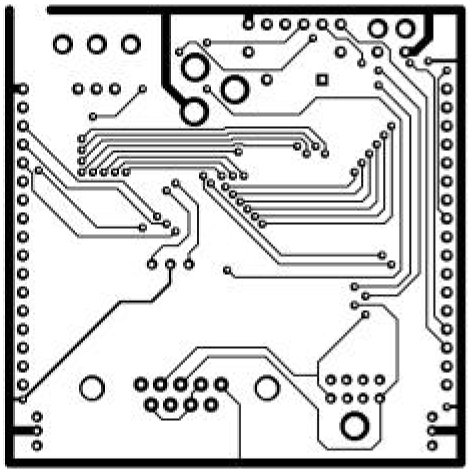


Figure 7: Serial Interface Block Diagram

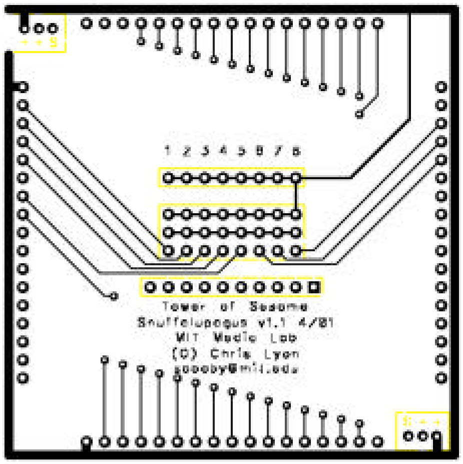
The Serial Interface's actual layout consists of two PCB boards stacked on top of each other (utilizing header pins). One of the boards contains the PIC^{TM} and serial componenets, the other board contains the sensor ports. These two PCB layouts are shown on the following page.



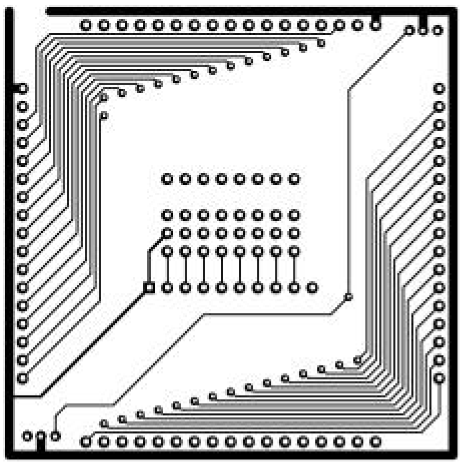
PIC^{TM} Board (top)



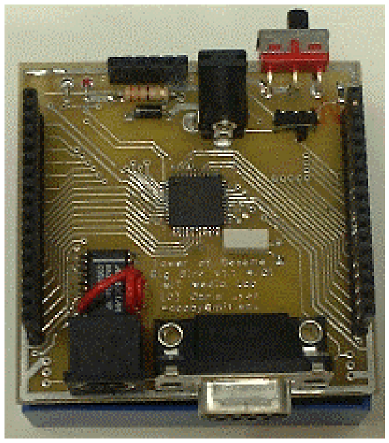
PIC^{TM} Board (bottom)



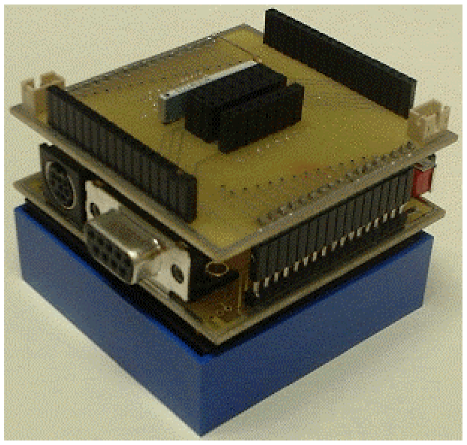
Sensor Board (top)



Sensor Board (bottom)



Assembled PIC^{TM} Board



Assembled Interface

Figure 8: Serial Interface PCB Layout

Software

The assembly code used on the PIC^{TM} will also be provided here. First however, the mapping[4] between the MicroChip PIC^{TM} op-codes and the op-codes used in the Cricket assembler are provided.

In general:

** The w register has been renamed **a**. This convention is followed by almost every other processor that only has a few registers.
** In the Cricket assembler the addressing mode is part of the op-code rather than an op-code/syntax combination. E.g. the add instruction are

```
add x
addn x
addm
```

The first form adds the contents of ram location x to a. The "n" form (read as "add number") adds a constant value to a. The "m" (read as "add to memory") form adds a to a memory location.

The inc instructions are

```
inc x
linc x
```

The first form increments memory location x. The "l" form (read as "load and increment") loads the contents of location x into a and then increments a.

The "mov" instructions have been renamed as load (lda) and store (sta) instructions - again following the model of most other simple CPU's.

** A few other instructions (e.g. goto, call > bra, bsr) have been renamed.

** The order of the inputs to the bit instructions has been swapped.

addwf x,0	>>	add x
addwf x,1	>>	addm x
addlw x	>>	addn x
andwf x,0	>>	and x
andwf x,1	>>	andm x
addlw x	>>	andn x
iorwf x,0	>>	or x
iorwf x,1	>>	orm x
iorlw x	>>	orn x
subwf x,0	>>	sub x
subwf x,1	>>	subm x
sublw x	>>	subn x
xorwf x,0	>>	xor x
xorwf x,1	>>	xorm x
xorlw x	>>	xorn x
comf x,0	>>	lcom x
comf x,1	>>	com x
decf x,0	>>	ldec x
decf x,1	>>	dec x
decfsz x,0	>>	ldecsz x
decfsz x,1	>>	decsz x
incf x,0	>>	linc x
incf x,1	>>	inc x
incfsz x,0	>>	lincsz x
incfsz x,1	>>	incsz x
rlf x,0	>>	lrol x
rlf x,1	>>	rol x
rrf x,0	>>	lror x
rrf x,1	>>	ror x
swapf x,0	>>	lswap x
swapf x,1	>>	swap x
movf x,0	>>	lda x
movf x,1	>>	tst x

movwf x	>>	sta x
movlw x	>>	ldan x
clrf x	>>	clr x
clrw	>>	clra
bcf x,b	>>	bclr b x
bsf x,b	>>	bset b x
btfsc x,b	>>	btsc b x
btfss x,b	>>	btss b x
call x	>>	bsr x
goto x	>>	bra x
retfie	>>	rti
return	>>	rts
retlw k	>>	rtv k
nop	>>	nop
clrwdt	>>	clrwdt
sleep	>>	sleep

We now provide the PIC^{TM} code:

```
; PocketPC Interface
; Sam Thibault
; samt@mit.edu
; Chris Lyon
; scooby@mit.edu
; 3-1-01

; watch out for $0c as temp register
; it is used for serial comm

; "hardware" registers
      [const @ 0]
      [const timer 1]
      [const pcl 2]
      [const status 3][const c 0][const z 2] [const bankl 5] [const
bankh 6]
      [const @@ 4]
      [const porta 5]
      [const portb 6][const portb-ddr $86]
      [const portc 7]
      [const portd 8]
      [const porte 9]
      [const pir1 $0C][const rcif 5]
      [const rcsta $18][const spen 7][const cren 4][const oerr 1]
      [const txreg $19]
      [const rcreg $1A]
      [const t1 $20]
      [const transmit-byte $21]
      [const receive-byte $22]
      [const sensor-reading $23]
      [const bus-code $24]
      [const led-port portb]
      [const led-pin 4]
      [const option 1]      ; bank 1
      [const txsta $18][const csrc 7][const txen 5][const sync 4][const
brgh 2]
      [const spbrg $19]
      [const tx-port portb][const tx 1]

      [const adresh $1e]
      [const adcon0 $1f] ;bank zero
      [const adcon1 $1f] ;bank one
      [const adon 0][const adgo 2]

      [const intcon $0b]
      [const gie 7]
      [const counter $25] ;$0c      ; bit counter for byte in process
      [const bus-data $21]
      [const bus-port portb][const bus 2]
      [const bus-port-ddr portb-ddr]


start
        [bsr io-init]

loop
        [bsr serial-tyi]
      [bra dispatch]

dispatch
      [btsc 7 receive-byte]
      [bra bus-dispatch]
      [btsc 6 receive-byte]
      [bra midi-dispatch]
      [bra sensor-dispatch]
        [bra loop]

bus-dispatch
      [lda receive-byte]
      [sta bus-code]

      [btsc 0 bus-code] ; send 1 byte
      [bsr bus-disp-send1]
      [btsc 1 bus-code] ; send 2 bytes
      [bsr bus-disp-send2]
      [btsc 2 bus-code] ; send 4 bytes
      [bsr bus-disp-send4]

      [btsc 3 bus-code] ; receice 1 byte
      [bsr bus-disp-rec1]
      [btsc 4 bus-code] ; receive 2 bytes
      [bsr bus-disp-rec2]
      [btsc 5 bus-code] ; receive 4 bytes
      [bsr bus-disp-rec4]

      [bra loop]

bus-disp-send1
      [bsr serial-tyi]
      [lda receive-byte]
      [sta transmit-byte]
      [bsr bus-tyo]
      [rts]

bus-disp-send2
      [bsr bus-disp-send1]
      [bsr bus-disp-send1]
      [rts]

bus-disp-send4
      [bsr bus-disp-send2]
      [bsr bus-disp-send2]
      [rts]

bus-disp-rec1
      [bsr bus-tyi]
      [lda bus-data]
      [sta transmit-byte]
      [bsr serial-tyo]
      [rts]

bus-disp-rec2
      [bsr bus-disp-rec1]
      [bsr bus-disp-rec1]
      [rts]

bus-disp-rec4
      [bsr bus-disp-rec2]
      [bsr bus-disp-rec2]
      [rts]

sensor-dispatch
      ;full return not implemented but individual readings are for 0-4
      [btsc 2 receive-byte] ;sensor 4
      [bra read-sensore]
      [btsc 1 receive-byte] ;sensor 2 or 3
      [bra sensor-dispatch2]
      [btsc 0 receive-byte] ;sensor 0 or 1
      [bra read-sensorb]
      [bra read-sensora]
sensor-dispatch2
      [btsc 0 receive-byte]
      [bra read-sensord]
      [bra read-sensorc]

sensor-return
      [lda sensor-reading]
      [sta transmit-byte]
      [bsr serial-tyo]
      [bra loop]
```

```
read-sensora
      [ldan 1] ;chselect0 + adon
      [bsr get-sensor]
      [bra sensor-return]

read-sensorb
      [ldan 9] ;chselect1 + adon
      [bsr get-sensor]
      [bra sensor-return]

read-sensorc
      [ldan $11] ;chselect2 + adon
      [bsr get-sensor]
      [bra sensor-return]

read-sensord
      [ldan $19] ;chselect3 + adon
      [bsr get-sensor]
      [bra sensor-return]

read-sensore
      [ldan $21] ;chselect4 + adon
      [bsr get-sensor]
      [bra sensor-return]

get-sensor
      [sta adcon0]            ; turn on converter
      [ldan $19] [bsr delay-loop]   ; wait 100 microsec acquisition
time
      [bset adgo adcon0]            ; start the conversion
sens20      [btsc adgo adcon0][bra sens20]      ; wait until done
      [lda adresh][sta sensor-reading]    ; read and return the result
      [rts]

midi-dispatch
      [btsc 0 receive-byte]
      [bra midi2-dispatch]
      [bra midi3-dispatch]

midi2-dispatch
      [bsr serial-tyi]
      [lda receive-byte]
      [bsr midi-out]

      [bsr serial-tyi]
      [lda receive-byte]
      [bsr midi-out]

      [bra loop]

midi3-dispatch
      [bsr serial-tyi]
      [lda receive-byte]
      [bsr midi-out]

      [bsr serial-tyi]
      [lda receive-byte]
      [bsr midi-out]

      [bsr serial-tyi]
      [lda receive-byte]
      [bsr midi-out]

      [bra loop]

serial-tyi
        [clr rcreg]
        [bclr rcif pir1]        ; clear the flag bit
        [bset cren rcsta]
s-tyi-mid
        [btss rcif pir1][bra s-tyi-mid]
        ; read is complete
        [lda rcreg]
        [sta receive-byte]
        [clr rcreg]
        [bclr oerr rcsta]
        [rts]

serial-tyo
        [bset bankl status]
        [bset txen txsta]
        [bclr bankl status]
        [lda transmit-byte][sta txreg]
        [bset cren rcsta]
        [rts]

midi-out
      [sta t1]
      [bclr tx tx-port]
      [ldan 14][bsr delay-loop]
      [bsr so][bsr so][bsr so][bsr so]
      [bsr so][bsr so][bsr so][bsr so]
      [bset tx tx-port]
      [ldan 14][bra delay-loop]

so    [ror t1]
      [bclr tx tx-port]
      [btsc 0 status]
      [bset tx tx-port] [nop] [nop] [nop]
      [ldan 13]
delay-loop
      [addn -1][btss z status][bra delay-loop]
      [rts]

io-init
        [bclr bankh status]

        [bset bankl status]     ;select bank 1

        ; Init baud rate for USART Synchonous master
        ;[ldan 25][sta spbrg] ; 9600 baud on 4MHz xtal
      [ldan 51] [sta spbrg] ; 9600 baud on 8MHz xtal
```

```
        ; Enable port
        [bclr sync txsta][bset brgh txsta]

      [ldan $00] [sta adcon1] ; set ra0,ra1,ra2,ra3,ra4 to analog

        [bclr bankl status]
        [bset spen rcsta]
        [bset bankl status]

        [bset csrc txsta]

      [bclr tx tx-port]

      [bclr led-pin led-port]
      [bclr bus bus-port]

        [bclr bankl status]     ; back to bank 0

      [bset bus bus-port]
      [bset tx tx-port]

        [clr transmit-byte]
        [clr receive-byte]

      ; set to read port?
      [ldan bus-port-ddr][sta @@]
      [bclr bus bus-port]
      [bclr bus @]
      ;[bset bus bus-port]
      [bset bus @]

        [rts]

; The following subroutine is the 8MHz version of the standard bus
; data receiving subroutine.  The form of a byte is 100 usec low time
; (to allow for interupt latency) start bit(1) + 8 data bits + stop
; bit (0 for cmnd 1 for data).  Each bit is exactly 10 usec long, but
; the value is valid only for the last eight microseconds.  This code
; samples the bits between 3 and 5 us into each bit.  This allows for
; an interrupt to delay the routine by up to 4.5us and still allow the
; data to be received.  The subroutine was tested with delays of 9
; instruction cycles and removal of one delay instruction; if the delay
; is 10 instructions or two delay instructions are removed, the data
; may not be received correctly.  The data is returned in bus-data.
; The inverse of the stop bit is returnted in the carry bit
; commands have a 0 stop bit -> carry set
; data has a 1 stop bit -> carry clear
; 21 July 1999   Jan Malasek

bus-tyi     ;[bset led-pin led-port]
      [btsc bus bus-port][bra bus-tyi]
btyi20      [btss bus bus-port][bra btyi20]     ; for sync edge
      [ldan 8][sta counter]
      [bsr an-rts]
      [bsr an-rts]
      [bsr an-rts]
      [bset led-pin led-port]; [nop]
```

```
btyi30      [nop][nop][nop]
      [nop][nop][nop]
      [ror bus-data]
      [bclr 7 bus-data]
      [btsc bus bus-port]
      [bset 7 bus-data]
      [bsr an-rts]
      [nop][nop][nop]
      [decsz counter]
      [bra btyi30]
      [bsr an-rts]
      [bsr an-rts]
      [bclr led-pin led-port] ; <-- test thingy
      [bset c status]
      [btsc bus bus-port][bclr c status]  ; no stop bit -> carry clear
an-rts      [rts]

bus-tyo
      ; set to write port
      [ldan bus-port-ddr][sta @@]
      [bclr bus bus-port]
      [bclr bus @]

      ; go low for 100 us sync time
      [bclr bus bus-port]
      [ldan 50] [bsr delay-loop]

      ; send start bit
      [bset bus bus-port] [nop]
      [bsr bus-delay]
      [bclr bus bus-port]

      ; send bit 0-7
      [btsc 0 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      [btsc 1 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      [btsc 2 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      [btsc 3 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      [btsc 4 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]
      [btsc 5 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      [btsc 6 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      [btsc 7 transmit-byte]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      ; command bit?
      [btss 7 bus-code]
      [bset bus bus-port]
      [bsr bus-delay]
      [bclr bus bus-port]

      ; stop bit
      [nop] [nop] [nop]
      [bsr bus-delay]

      ; command bit maintanence
      [bclr 7 bus-code]

      ; set to read port
      [bset bus @]

      [rts]

bus-delay ; to get 10 us bits on bus line
      [ldan 2]
      [bsr delay-loop] [nop]
      [rts]
```

Appendix C: MICK User Manual

This appendix provides a brief instruction manual for using MICK with descriptions of the many effects that can be produced.

Getting Started. When MICK is started a screen with a large logo will appear. Before starting an instrument it is a good idea to test that the Serial Interface and MIDI are working properly. To do this, select *Test Midi* from the *Instrument* menu. You should then hear a note played through the MIDI. If this does not work, check that power is supplied and that the interface is connected to the PocketPC^{TM}.

Now, select *New* from the *File* menu. (Note: you could also open a previously saved file by using the *Open* command.) When you start this new instrument a dialog box will appear asking what type of sensor you would like to add as the first sensor in your instrument. Choose the appropriate sensor from the options displayed. Once the first sensor has been selected, the main screen will display the sensor as an icon, with associated options and information displayed in a row across the screen. The icons that represent the different sensors are:

Touch	*Light*	*Temperature*	*Distance*

Setting the Location

To set the location of a sensor, click on the gray box immediately to the right of the sensor icon. A dialog box will appear where the sensor's location can be specified in terms of bus location and port number. For bus device sensors like a distance sensor, leave the port field set to 0.

Setting the MIDI voice and channel

The next box to the right of the location box allows the MIDI voice associated with the sensor to be selected. Clicking on that icon will open a dialog box in which you can set the channel and voice for the instrument. Note that modifying the voice used by some channel will affect all sensors that use that channel.

Deleting a Sensor

On the far right of the screen is a box with an "X". Clicking on this box will remove the sensor and all its actions from the instrument. To add a new sensor, select *Add Sensor* from the *Instrument* menu.

Adding Sensor Actions. The remaining icons that have a "+" symbol contained inside them allow you to add actions and musical effects to the sensor. Clicking on one of these icons will open a dialog box that allows the action to be specified in detail.

Playing Notes and Chords

The first action allows for the playback of notes or chords. When the dialog box opens there are two main sections. At the top are boxes that allow the range to be specified. Use these boxes to specify the values inside which this action to occur. For example, if the range is from 100 to 200, the note/chord will be played if the sensor's value is in that range. To add a note to be played click the *add note* button. To move between notes click on the *next* button. A selected note can be removed by clicking the *delete note* button. To modify the pitch of the selected note click on *raise pitch* or *lower pitch*. Finally, at the bottom of the dialog is a check box where the root pitch can be played. The root pitch is a special note that can be modified by other actions. The shift pitch action described next will talk more about this effect in detail. When finished modifying the note, click the *ok* button in the top right corner of the screen.

Once you leave the dialog box a new row of icons will appear indented underneath the sensor you added the action. This indention will allow you to identify which actions are associated with what sensors. To modify the actions later click on the icon at the left of the actions row.

Shifting the Root Pitch

The next action that can be added (represented with a sliding note) allows the root pitch to be modified. Again the first part of the dialog box allows the range for this action to occur. The rest of the dialog allows the direction and amount of the shift to be specified. The root pitch itself is specified from the selection *Root pitch* in the *Instrument* menu.

Playing Wave Files

The next action allows a wave file to be specified for playback. Again enter the range, as well as the name of the wave file to play back. The wave file will be played through the internal speaker in the PocketPC^{TM}.

Activating Motors, Displays, and Tri-color LEDs

The three actions after playing wave file are all used for non-musical output. These actions allow the interface to activate motors, displays, and tri-color leds, respectively. In these dialog boxes, as the others, select the range for the action to occur first. Next, since all these devices are bus devices, select their bus location. Finally the specifics of the action, either motor direction or display detail, are entered. Again, click *ok* to finish.

Executing Scripting Commands

The icon with a turtle is intended for future use. It will allow scripting commands to be specified.

Performing Functional Mappings

The final action (picturing a graph) allows mathematical mappings to be specified. After selecting the range, select the type of function to use. The two values *m* and *b* will be used as described in the table 3. At the bottom of the screen it is possible to select the type of outputs. You may choose from note, volume, display, and led mappings.

Function	**Relationship to *m* and *b***
Linear	$y = m * x + b$
Inverse	$y = m / x + b$
Square	$y = m * x^2 + b$
Square Root	$y = m * \mathrm{sqrt}(x) + b$

Table 3: Mapping Functions

Playback and Finishing Up. Once the instrument has been set up with sensors and actions its operation can be started by selecting *Run* from the *Instrument* menu. The instrument can also be saved using the options under the *File* menu. It is likely that an instrument will have more sensors and actions specified than will fit on the screen of the PocketPC^{TM}. If this occurs, use the *scroll up* and *scroll down* buttons to move the sensors and actions that are currently displayed.

Appendix D: MICK Scripting Primitives

In order to allow a series of actions to be performed on a single sensor event, MICK could allow users to program in a logo-like language. This language can perform all the state-setting actions that MICK provides, such as playing a note or turning on a motor, in a series of commands that may include timing delays, repeated sequences, or other control structures. Procedures may be declared as well. A sample set of instructions is given in table 3 below.

In order to allow sensor events to be able to trigger several interleaving series of MICK-Script sequences it is necessary to implement some form of multi-threading. To accomplish this objective, each sensor event is given its own queue for scripting commands. When that sensor event is triggered, the commands the user has connected with that event are added to that sensor event's queue. On each pass over the sensors while the instrument is running, each queue executes one instruction. To help avoid memory overflows due to recursive calls, instructions could be executed in a lazy manner. That is, a call to a procedure will not be expanded on the queue until that procedure call is reached. Notice, however, that MICK does not provide any strong assurances on the safety of the command sequences. If one thread sets a motor and then a different thread sets the motor to a different value. The most recent setting will win. Therefore, users must use care in using such a scripting feature.

Scripting Primitive	**Explanation**
To *proc-name (:arg1 :arg2 …)* *body* **end**	Creates a procedure named *proc-name.* This procedure may receive arguments named by variables starting with colons in the declaration. The body of the procedure is included before the **end** keyword.
Repeat *num* [*body*]	Repeats instructions in body of repeat statement *num* times.
if *cond* [*body*]	Tests the condition *cond*. If the value of *cond* is true, *body* is executed.
Ifelse *cond* [*body1*] [*body2*]	Tests the condition *cond.* If the value of *cond* is true, *body1* is executed, otherwise *body2* is executed.
Wait *num*	Does not perform an instruction for *num* instructions. (skips instruction cycles)
Display *num*	Displays number *num* on display bus device.
Sensor0	Returns the value of sensor in port 0.
Sensor1 … **sensor7**	Returns the value of sensor in port 1 - 7.
Sensor *color port*	Returns the value of sensor in port *port* on color *color* sensor port bus device.
Motor *color port direction power*	Sets motor in port *port* on color *color* motor bus device to direction *direction* and power *power*. *direction* is 0 or 1. *power* is 0-7
Note-on ***pitch channel***	…
Melody ***filename***	…
Clear-thread	Clear this sensor-event's thread
+ - * /	Basic mathematical operators.
etc.	etc.

Table 4: MICK Scripting Primitives

Appendix E: MIDI Instruments and Effects

This appendix provides tables of the instruments available on the standard MIDI channels (1-9 and 11-16), as well as a listing of the drum sounds for channel 10, which is restricted to those drum events, followed by descriptions of the variety of MIDI effects.

PC#	Instrument name	PC#	Instrument name	PC#	Instrument name	PC#	Instrument name
1	Acoustic Grand Piano	33	Acoustic Bass	65	Soprano Sax	97	FX 1 (rain)
2	Bright Acoustic Piano	34	Electric Bass (finger)	66	Alto Sax	98	FX 2 (soundtrack)
3	Electric Grand Piano	35	Electric Bass (pick)	67	Tenor Sax	99	FX 3 (crystal)
4	Honky-tonk Piano	36	Fretless Bass	68	Baritone Sax	100	FX 4 (atmosphere)
5	Electric Piano 1	37	Slap Bass 1	69	Oboe	101	FX 5 (brightness)
6	Electric Piano 2	38	Slap Bass 2	70	English Horn	102	FX 6 (goblins)
7	Harpsichord	39	Synth Bass 1	71	Bassoon	103	FX 7 (echoes)
8	Clavi	40	Synth Bass 2	72	Clarinet	104	FX 8 (sci-fi)
9	Celesta	41	Violin	73	Piccolo	105	Sitar
10	Glockenspiel	42	Viola	74	Flute	106	Banjo
11	Music Box	43	Cello	75	Recorder	107	Shamisen
12	Vibraphone	44	Contrabass	76	Pan Flute	108	Koto
13	Marimba	45	Tremolo Strings	77	Blown Bottle	109	Kalimba
14	Xylophone	46	Pizzicato Strings	78	Shakuhachi	110	Bag Pipe
15	Tubular Bells	47	Orchestral Harp	79	Whistle	111	Fiddle
16	Dulcimer	48	Timpani	80	Ocarina	112	Shanai
17	Drawbar Organ	49	String Ensemble 1	81	Lead 1 (square)	113	Tinkle Bell
18	Percussive Organ	50	String Ensemble 2	82	Lead 2 (sawtooth)	114	Agogo
19	Rock Organ	51	Synth Strings 1	83	Lead 3 (calliope)	115	Steel Drums
20	Church Organ	52	Synth Strings 2	84	Lead 4 (chiff)	116	Woodblock
21	Reed Organ	53	Choir Aahs	85	Lead 5 (charang)	117	Taiko Drum
22	Accordion	54	Voice Oohs	86	Lead 6 (voice)	118	Melodic Tom
23	Harmonica	55	Synth Voice	87	Lead 7 (fifths)	119	Synth Drum
24	Tango Accordian	56	Orchestra Hit	88	Lead 8 (bass + lead)	120	Reverse Cymbal
25	Acoustic Guitar (nylon)	57	Trumpet	89	Pad 1 (new age)	121	Guitar Fret Noise
26	Acoustic Guitar (steel)	58	Trombone	90	Pad 2 (warm)	122	Breath Noise
27	Electric Guitar (jazz)	59	Tuba	91	Pad 3 (polysynth)	123	Seashore
28	Electric Guitar (clean)	60	Muted Trumpet	92	Pad 4 (choir)	124	Bird Tweet
29	Electric Guitar (muted)	61	French Horn	93	Pad 5 (bowed)	125	Telephone Ring
30	Overdriven Guitar	62	Brass Section	94	Pad 6 (metallic)	126	Helicopter
31	Distortion Guitar	63	Synth Brass 1	95	Pad 7 (halo)	127	Applause
32	Guitar Harmonics	64	Synth Brass 2	96	Pad 8 (sweep)	128	Gunshot

Table 5: General MIDI Melodic Voices

Note	Drum Sound	Note	Drum Sound
B0	Acoustic Bass Drum	B2	Ride Cymbal 2
C1	Bass Drum 1	Middle C (C3)	Hi Bongo
C#1	Side Stick	C#3	Low Bongo
D1	Acoustic Snare	D3	Mute Hi Conga
Eb1	Hand Clap	Eb3	Open Hi Conga
E1	Electronic Snare	E3	Low Conga
F1	Low Floor Tom	F3	High Timbale
F#1	Closed Hi-Hat 1	F#3	Low Timbale
G1	High Floor Tom	G3	High Agogo
Ab1	Pedal Hi-Hat 1	Ab3	Low Agogo
A1	Low Tom	A3	Cabasa
Bb1	Open Hi-Hat 1	Bb3	Maracas
B1	Low Mid Tom	B3	Short Whistle
C2	Hi Mid Tom	C4	Long Whistle
C#2	Crash Cymbal 1	C#4	Short Guiro
D2	High Tom	D4	Long Guiro
Eb2	Ride Cymbal 1	Eb4	Claves
E2	Chinese Cymbal	E4	Hi Wood Block
F2	Ride Bell	F4	Low Wood Block
F#2	Tambourine	F#4	Mute Cuica
G2	Splash Cymbal	G4	Open Cuica
Ab2	Cowbell	Ab4	Mute Triangle
A2	Crash Cymbal 2	A4	Open Triangle
Bb2	Vibraslap		

Table 6: General MIDI Percussion Sounds (channel 10)

Effect	Description
Modulation Wheel	Used to control pitch modulation (vibrato) level on a specified channel
Volume	Used (in conjunction with Expression) to control overall volume of notes on a specific channel
Pan	Used to control left/right output placement for notes on specified channel
Expression	Used (in conjunction with Volume) to control overall volume of notes on a specific channel
Damper Pedal (Sustain)	Allows notes on a specified channel to continue sounding after the corresponding notes have been released. Notes are terminated when damper pedal is turned off.
Sostenuto	Similar to Damper Pedal, except Sostenuto only effects note which were already active when Sostenuto is turned on.
Effect 1 Depth (Reverb)	Used to adjust the amount of reverb effect applied to sounds played on a specific channel
Effect 2 Depth (Chorus)	Used to adjust the amount of chorus effect applied to sounds played on a specific channel

Table 7: General MIDI Effects