Microcontrollers in Music HCI Instruction: Reflections on our Switch to the Atmel AVR Platform

Scott Wilson
CCRMA, Department of Music
Stanford University
Stanford, California USA
rswilson@ccrma.stanford.edu

Michael Gurevich
CCRMA, Department of Music
Stanford University
Stanford, California USA
gurevich@ccrma.stanford.edu

Bill Verplank
CCRMA, Department of Music
Stanford University
Stanford, California USA
verplank@ccrma.stanford.edu

Pascal Stang
Department of Electrical Engineering
Stanford University
Stanford, California USA
pstang@stanford.edu

ABSTRACT

Over the past year the instructors of the Human Computer Interaction courses at CCRMA have undertaken a technology shift to a much more powerful teaching platform. We describe the technical features of the new Atmel AVR based platform, contrasting it with the Parallax BASIC Stamp platform used in the past. The successes and failures of the new platform are considered, and some student project success stories described.

Keywords

Microcontrollers, Music Controllers, Pedagogy, Atmel AVR, BASIC Stamp.

1. INTRODUCTION

Every year since 1996, CCRMA has offered a course focusing on human-computer interaction. After the early versions of the course which involved teleconferencing with San Jose State and Princeton, it became Stanford's "course on controllers", offering a hands-on approach to interaction design for musical applications. Until this past year, the core technology was Parallax's BASIC Stamp BS2SX[6] in conjunction with an analog-to-digital converter (ADC) such as Maxim's Max1270[4] and a variety of sensors[11]. In the summer of 2002 CCRMA introduced a new workshop: Physical Interaction Design for Music (PIDM), a two-week summer course using a new technology platform based on the Atmel AVR microcontroller[1]. That platform was subsequently upgraded and further developed for the controllers course in the fall. The switch has proved largely successful, providing many advantages over the previous platform, and resulting greatly improved student work. It is the authors' hope that this paper will facilitate a dialogue among educators to discuss and assess the merits of different technologies being used for teaching in this field. Furthermore, we hope that this paper will provide motivation for others to frankly evaluate current technologies and make appropriate changes where they are needed.

In both the PIDM and HCI courses, a series of simple exercises are done to introduce sensors, signal-conditioning, microcontroller programming, communication, music programming. In addition, they are all done in the context of measuring human performance: latency, repetition-rate, logarithmic thresholds. Finally, a framework is presented for organizing a user-interface design project. All of these are reported in our first NIME paper [11].

2. OVERVIEW OF MICROCONTROLLERS

The AVR series of microcontrollers is a recent addition to the field of 8-bit microcontrollers. Table 1 compares several of the most commonly used microprocessor technologies. The Intel 8051- and Motorola HC11-based processors have long histories, numerous derivatives in the case of the 8051, and are widely used in embedded systems. The PIC and AVR series are more recent technologies that have also become firmly established in embedded systems. The AVR line has several key advantages over the PIC line with which it is clearly designed to compete. Most importantly, the AVR core is 4 time faster than the PIC due to its remarkable single clock cycle instruction execution speed. The AVR core was also designed with attention to the requirements of C compilers. As a result, it has strong compiler support from Atmel and the maturation of its open source compiler and tool chain has been much faster and has far surpassed that of PIC. The AVR's ISP (In System Programming) support saves time and reduces damage to chips by minimizing the number of times they are inserted and removed from a circuit.

All of the processors in Table 1 belong to large families of chips with varying features built into each of them. All have variants with on-board A/D conversion, SPI, I2C and UART support. The HC11 is notably supported as a compile target in Metrowerks CodeWarrior, a widely used cross-platform compiler for embedded systems and main-stream operating systems.

The BASIC Stamp is fundamentally different from these processors in that it embeds a microprocessor and its instruction set is entirely emulated. Therefore, it has been omitted from the table, but it will be discussed in detail in the following section.

	AVR	PIC	Intel 8051	Motorola HC11
Architecture	Harvard	Harvard	Harvard	Von Neumann
Instruction Set	RISC	RISC	CISC	CISC
Avg. Clock Cycles per Instruction	1	4	24	16
Std. Clock Speeds	8,16Mhz	10,20,40Mhz	12Mhz	8Mhz
MIPS	8,16	2.5,5,10	0.5	0.5
Memory Technologies	flash	flash,OTP	flash,OTP EEPROM,ROMless	flash,OTP EEPROM,ROMless
Typ. Program ROM	16KB	4KB	4KB	8KB
Accumulator	No	Yes	Yes	Yes
gcc support	Yes	No	No	Yes

Table 1: Comparison of commonly used microprocessors.

3. PREVIOUS PLATFORM - PARALLAX BASIC STAMP

The BASIC Stamp[6] is an excellent teaching tool and is certainly the most self-contained microcontroller platform on the market. The typical BASIC Stamp[1] comes in a 20-pin DIP package with 16 general purpose digital I/O pins. The user programs the chip in BASIC on a Windows PC using Parallax's compiler and monitor software. The Stamp's EEPROM program memory is then programmed via an RS-232 serial connection. The serial connection also serves to transmit run-time data and debug messages back to the monitor software on the PC.

The Stamp BASIC interpreter provides a set of special purpose commands in addition to a minimal BASIC language implementation. Notable feature examples include serial input and output on any pin, X10 lighting device control, pulse width modulation output and RC filter time constant measurement.

Two critical weaknesses of Stamp BASIC are the inability of the multiplication and division operators to handle negative numbers, and the lack of floating point support. Fixed point operations which are fundamental to C and assembly programs are also costly to implement in Stamp BASIC.[2]

The processor executes at a slow rate of approximately 4,000 instructions per second. This slow execution time is the most evident result of the overhead involved in providing such a friendly onboard interpreter. For many applications such as traditional buttons and display interfaces, simple robotics, and low-rate data acquisition and reporting this execution rate is not a problem. However, as students refined and attempted to scale their projects or strove for optimized low-latency performance, the speed of the Stamp became a barrier to further progress. Unfortunately, there is no easy way to upgrade, no incremental step up to a more powerful or better suited processor in the Stamp line without significantly revising a student's initial work investment.

We encountered several problems in using the Stamp as the platform for our courses. Particularly in our academic courses, fewer and fewer students have previous experience with BASIC. Many of them come in with existing knowledge of C and / or a higher level language such as Java. For these students learning BASIC is frustrating because it is very limited, has a more confusing syntax, and violates many of the stylistic rules they have already learned.

The lack of flexible encapsulation mechanisms in BASIC affects our ability to provide and refine program building blocks for the students. BASIC encourages monolithic programs and cut and paste coding whereas the C paradigm is functional and more structured.

The extremely small RAM and ROM memory space available on Stamp processors is also a limiting factor. Even simple applications can exceed the available memory resources of RAM and ROM.

Another significant downside to the Stamp is in the cost per chip. At approximately 60$ US per unit, costs mounted quickly given that chips were getting accidentally burned out at a rate of about one a week. The final serious drawback to the Stamp platform was the lack of a Linux version of the programming software. In our primarily Linux-equipped lab this was a serious bottleneck during lab sessions[3].

4. CURRENT PLATFORM - ATMEL AVR

4.1 Hardware

Our current hardware platform is based on the Atmel AVR ATmega163 8-bit RISC microcontroller[4]. We use the AVR on a custom-designed small-footprint board[10] and program it from Linux and Windows platforms, with Macintosh OS X programming also possible. The AVR series has a large active user base ranging from professional embedded-systems designers to hobbyists. One of the most important products of that community has been the AVR's inclusion as a standard build target in the open source Gnu *gcc* compiler.

4.1.1 The Processor

The AVR ATmega163 microcontroller has a maximum clock speed of 8MHz and the majority of its instructions

[1]Our most recent teaching experience with the BASIC Stamp was with the BS2SX. Parallax has since released two relevant chips, the BS2P and the Javelin stamp. The BS2P is reported to be 20% faster, has an optional 40-pin package doubling the number of available I/O pins, interfaces directly to parallel LCD modules, I2C chips, One Wire chips, and supports interrupt-driven programming. The Javelin stamp is programmed in a subset of the Java programming language and has significantly larger RAM and program EEPROM, both essential to support the higher level language.

[2]These problems have been partially addressed in the BS2SX and BS2P series.

[3]At the time of writing, Parallax has finally released a Linux and Macintosh-supported development library for creating IDEs on those platforms.

[4]At the time of writing the ATmega163 has been replaced by the pin-compatible ATmega16 which supports clock speeds of up to 16MHz, the low power ATmega16L now has a maximum speed of 8MHz.

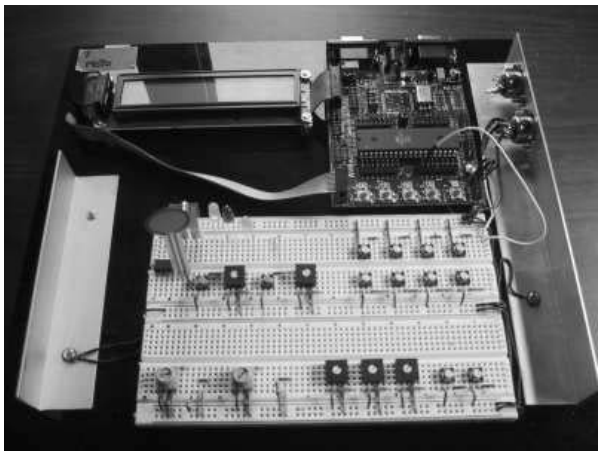

Figure 1: Prototyping Kit

complete in a single cycle thus providing assembly code performance near 8 MIPS. Translated into C instructions, we can estimate the execution at a factor of ten slower, still significantly faster than the speed of the Stamp.

The ATmega163 has 16KB of flash program memory, 1KB of data SRAM, and 512 bytes of EEPROM memory[5]. The processor has several interrupt sources so the program may respond to a number of internally and externally-generated events such as timer overflows and the completion of ADC or serial communication operations. The AVRlib[9] function library allows us to package the handling of these interrupts in a straightforward manner which the students quickly learn to use.

Standard microcontroller features found on most of the ATmega series of AVR microcontrollers include an interrupt-controlled UART for serial communication and three independant hardware timers, one of which can be synchronized to an external Real Time Clock (RTC) oscillator. The RTC clock facilitates developing programs that deal in minutes, seconds, and fractions of seconds, useful in many music applications. The processor also supports up to three channels of Pulse Width Modulation (PWM) output. The ATmega processors support the I2C and SPI communication protocols facilitating the addition of external peripheral ICs such as EEPROMs, programmable logic devices (PLDs) and digital to analog converters (DACs).

We chose the AVR series of processors for the cost, memory size, speed, and availability of linux supported development tools. Specifically, we selected the ATmega163 for its eight channels of integrated 10-bit analog to digital conversion (ADC). Having integrated ADCs simplifies the task of reading continuous sensor circuits. The processor provides several choices of analog voltage reference for the ADC including an external reference, simplifying the task of scaling the sensor signals. The conversions complete in as little as 65μs, amply fast for the design of low latency physical interfaces. The ADC support was the most heavily used feature in many student projects.

4.1.2 The Development Board - AVRmini

We use a development board designed by Pascal Stang

[5]The flash memory stores the compiled program code, the SRAM holds run-time data, and the EEPROM is intended for storing calibration information or other data that must persist over power interruptions.

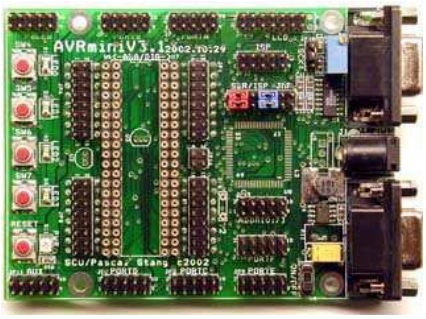

Figure 2: The AVRmini Development Board (top)

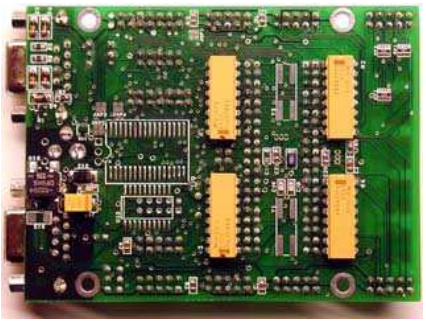

Figure 3: AVRmini (bottom)

called the AVRmini[10] to hold the ATmega163 microprocessor. The AVRmini provides convenient access to the I/O ports of the processor via blocks of headers. The headers enable individual pins or entire I/O ports to be connected to a wireless prototyping board using jumper cables. Sensors are also easily connected directly to the pins of the AVRmini using jumpers. The board has resistor packs protecting all of the I/O pins by default which can be selectively disabled by installing bypass jumpers. This simple addition protects the microprocessor from damage caused by incorrect wiring of the I/O pins. We haven't had any processors ruined this way yet, a much better record than we had using Stamp chips.

The AVRmini provides a header to connect an industry-standard character LCD module using four or eight I/O lines. The LCD modules are ubiquitous and inexpensive when purchased from surplus electronics shops. They are a valuable tool for displaying debugging information and for projects requiring state display. The AVRlib[9] library provides several convenient options for output to the LCD. The built-in set of four LEDs and four pushbuttons on the AVRmini may be connected via jumper cables to any of the I/O ports. This functionality has proven useful especially in the early stages of learning about the processor, the board and basic pushbutton and light circuits. The board provides an RTC clock crystal for clocking the third timer which can be enabled via jumpers.

Compiled code is downloaded to the microprocessor from a computer using an RS-232 serial connection. The AVRmini provides two RS-232 connectors. Either one can be patched to the UART so that the programming interface and a serial communications link may be kept connected simultaneously.

The AVRmini supports all 40-pin DIP and 64-pin QFP processors in the AVR series providing flexibility in choosing a processor with features suited to the specific application.

Provision is also made for a bank of external SRAM of up to 512KB. An efficient switching voltage regulator prolongs battery life for autonomous applications and a separate analog voltage reference regulator may also be installed on the board.

4.2 Software

We have used Linux almost exclusively for the programming and performance of the instruments created in these courses because CCRMA's systems are nearly all Linux-based. An IDE for the AVR exists for Windows, and thanks to the open source tool chain built around *gcc*[2], our development environment is programmable from all of the modern operating systems in use today.

4.2.1 Compiler and Loader

Programs are written in C with the addition of special purpose macros for the AVR. A makefile automates the process of compiling the AVR C code, linking it with the standard C library and the AVRlib library, generating the memory map and hex code files, and uploading them into the processor. The compiled program is uploaded to the processor using the command line utility *uisp*[7]. Using C enables us to provide the students a library of functions with a consistent interface. As a result, we are able enhance the library and fix bugs without disrupting the students' work flow. While both cut-and-paste code and function libraries are likely to contain code the student does not understand, they can be encouraged to read the code in the library without the danger of their introducing bugs into that part of the code as often happens with monolithic programs.

This is a good example of a case where we can simplify the process for students to make initial operation painless, while keeping everything accessible should they decide to dig deeper. A student can open up the makefile we provide and begin to customize it to suit their needs. By investigating what the makefile and compiler are doing, students can look into the disassembled source code, the memory map, and the hex code files that are generated from the compilation of their C code.

4.2.2 Support Library - AVRlib

Pascal Stang's AVRlib[9] library of C support routines is extensive and invaluable for work on this platform. AVRlib includes functions wrapping many of the standard features of the AVR processors. These include support for managing the timers, using the A/D converters, the UART, SPI and I2C interfaces, and the PWM outputs. General purpose code in the library provides bit and byte oriented data buffers, an implementation of a convenient C-style printf function, and terminal emulation facility. There is also support for specific peripherals including character and graphical LCD modules, external SRAM, GPS, ATA hard drives and an MP3 player! This monumental library continues to grow and improve as Pascal uses it in his own teaching and personal projects.

4.2.3 Control Output

In an effort to provide the students a flexible set of options for the output of their controllers we supported both MIDI and Open Sound Control (OSC)[5]. For MIDI, the standard MIDI output circuit was attached to the UART pin of the processor and wrapper functions for the UART library routines were provided to give the students functions for standard MIDI message types. OSC support was achieved by compiling the OSC message construction code and a similar set of wrappers around the UART library functions. The OSC messages were received on the Linux computers via RS-232 serial rather than a midi interface. The OSC UDP receiving object for Pd[8] was modified to create an object called OSCSerial that receives messages from the serial port rather than the network[13]. Here, again, students can send MIDI or OSC messages with a single C function call, but have access to the libraries to see exactly what is going on behind the scenes and customize these functions if desired.

In the two courses we have taught using this platform, Pd has been the software of choice for designing the musical side of the projects because of its availability and popularity at CCRMA. Several users have also used the MIDI output functionality to communicate with Max, Pd and commercial music gear.

4.3 Motivation for Switching

The aforementioned limitations of the Stamp platform and the lack of Linux software were the initial motivating factors for seeking a new teaching platform. As we considered options for the new platform we realized the benefits of choosing a platform with a stronger more structured programming approach, where C in particular provides a wealth of existing free code that can be ported, modified and appropriated at will. Using *gcc* as the compiler is also an advantage because it provides a coherent transition from programming C in Linux and Mac OS X to programming for the AVR. Equipment cost was another key factor in our switch as a fully outfitted AVRmini board with an ATmega163 processor costs roughly the same as a BS2P Stamp processor.

	Pros	**Cons**
BASIC Stamp	Thorough docs geared to students, self-contained platform, no dev. board needed, simple to program.	Costly, easily damaged, slow execution speed, small memory size, closed platform, Windows dependant.
Atmel AVR	Fast execution, low cost, large family, programmable in C, gcc / linux integration, on-board A/D.	Student must confront and understand a complex architecture, documentation sparse, written for engineers.

Table 2: Summary of platform pros and cons.

5. OUR EXPERIENCES

A recurring theme in our course has been "Why Microcontrollers?". In other words, why not give students a black box that does 8 channels of A/D conversion on 0-5V signals and generates MIDI messages? The short answer is pedagogical. Using a programmable microcontroller allows the students to learn about computer architecture, digital logic, programming, A/D conversion and serial and parallel communication protocols. In learning to program and use a microcontroller, students develop these skills and intuitions in a practical, hands-on way that would be difficult with theory alone. Furthermore, it gives students exposure to the technology used in actual commercial products, demystifying the world of embedded systems. In this respect, the switch to the AVR has been significant. It represents a shift from essentially a hobbyist's toolkit to professional, commercial-grade technology that is much closer to the technology used in a wide variety of existing commercial devices.

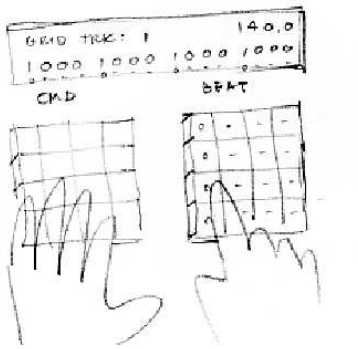

Figure 4: Beat Matrix

Admittedly, using a microcontroller, and specifically the AVR, creates more work for the students, though their ability to fully grasp the technology even in the context of our 2-week summer workshop[12] was impressive. The platform's infrastructure, including the AVRlib support libraries, the avr-gcc compiler, and makefiles help make the technology accessible for the novice student, but also importantly gives the advanced student full "under-the-hood" access to a set of well-established, industry standard tools to customize his or her work.

Beyond making students' lives difficult, microcontrollers have enabled students to produce unique, highly successful, innovative projects that would not otherwise be possible. The new platform has greatly improved these facilities, as evidenced in a number of impressive student projects. Example uses of microcontrollers have included parallel LCD interfaces, precise timing, interrupt programming, wireless communication, digital filtering, interpolation, multiplexed I/O and pulse width modulation.

5.1 Project Successes

5.1.1 Beat Matrix

The Beat Matrix, created by David Lowenfels and Gregor Hanuschak, is a MIDI drum sequencer using two 4x4 keypads in conjunction with our development board. The goal was to create a 4 beat sequencer, using one 4x4 grid to represent the sixteenth note subdivisions of the beats. This project's success lies in its self-contained nature, which was afforded by the microcontroller. The controls and display are completely integrated into the device, relying on a computer or synthesizer for sound generation only. Our platform's LCD is integral to the device, giving the user immediate visual feedback for the controls, allowing the user to navigate through drum tracks and displaying the state of the sequencer. David and Gregor learned to manage the microcontroller's internal memory to create sequences, and used interrupts to effectively program the controls, LCD display and MIDI communication to ensure the steady timing essential for a sequencer. The students intend to add features including external memory to store user presets, which should be relatively easy with the current platform and the AVRlib's existing I2C support.

5.1.2 Muggling

Pascal Stang, Jeff Bernstein and John McCarty developed their "Muggling" project using some of the important capabilities the platform provides. "Muggling" comes from "musical juggling", describing their intention to instrument

Figure 5: Muggling

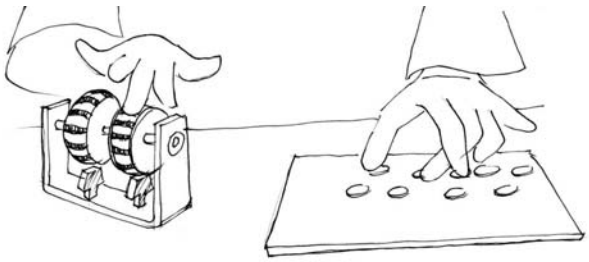

Figure 6: Rollerblade wheel encoder

3 juggling balls to send signals that could be used to control musical parameters. The group successfully implemented one ball as a remarkable proof of concept. The ball contains 4 two-axis accelerometers, from which linear and angular acceleration in the x, y and z planes are calculated. Analog acceleration values are sampled, low-pass filtered and interpolated to 14-bits on an AVR323, then transmitted via a Linx wireless radio transmitter to a Linx base station receiver.[3] The portability of the AVRlib code used in class allowed them to easily switch to a more capable microcontroller in the same family without any major changes. The technology's low cost and small package size options allowed the accelerometers, 3-color LED's, microcontroller and radio transmitter/receiver to be mounted inside a 3" diameter ball.

5.1.3 Rollerblade wheel encoder

The most successful project from our 2-week Physical Interaction Design for Music[12] workshop was a home-made optical encoder built by Chad Cosby. It consisted of a pair of Rollerblade wheels covered in alternating black and white stripes and mounted on a frame to sit on a desk. An inexpensive infrared LED and photodiode pair was attached beside each wheel to act as a light sensor, detecting changes from dark to light as the wheel spun. The LED/photodiode pair is commonly used in robotic devices to allow them to follow dark lines painted on the floor. By timing the transitions, Chad was able to measure the velocity of each wheel, which he encoded as a MIDI control signal and sent to a Linux computer to control sound synthesis parameters in Pd. The ability of the microcontroller to decode digital signals in this manner has also been used with commercial optical shaft encoders, and Duty Cycle Modulated signals from accelerometers.

5.1.4 Rings of Light

Sara Shaughnessy and Renee Goldschmid mounted the same LED/photodiode packages around the perimeter of two rings to function as proximity sensors when a performer places her hands inside the rings. Each ring contains three sensors, each controlling the intensity and timbre of a note of a chord. A third control ring and a series of force-sensitive pedals adjust the tuning and voicing of the chords. This

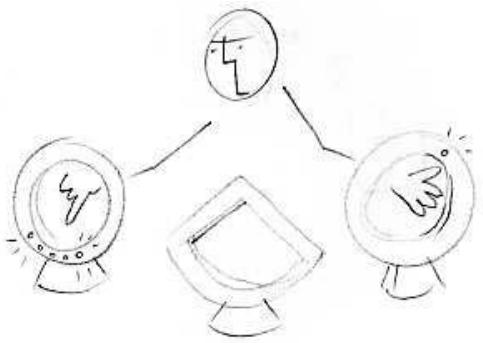

Figure 7: Rings of Light

project uses the AVR in combination with programmable logic devices to activate arrays of LEDs that are mounted above each sensor. These LEDs provide immediate feedback to the performer and a compelling display for the audience. Programmable logic and multiplexing were necessary in order to activate 30 LEDs from a limited number of digital output pins. Additionally, the analog inputs were multiplexed to take 12 continuous signals from the proximity sensors and foot pedals into the 8 A/D inputs of the AVR.

6. CONCLUSIONS

The switch to our current teaching platform has provided the students with a more fully-featured, robust and flexible system for lab exercises and projects. From a pedagogical standpoint, it has allowed us to deal more directly with microcontroller architecture and embedded systems design, as well as develop a library of functional code that is far more practical than a monolithic, cut-and-paste approach. Advanced students have access to powerful, professional-level tools that promote portability, scaleability, and ample support for refinement and true innovation in their projects. The new platform has the added advantages of easy integration in CCRMA's Linux environment and lower cost.

These improved capabilities have come at the expense of more overhead work on the parts of both the students and instructors. Having developed more support infrastructure and experience in teaching with the technology, it should be possible to use this platform for the PIDM workshop in the future with an improved student experience. Simplified tools for novice users will allow us to better focus on the different pedagogical objectives for a two-week workshop. In the controllers course, the quality of student work has improved dramatically since the introduction of the new platform. The system enabled students to create truly innovative projects employing a wide variety of sensing and communication technologies. The range of successful student projects has provided direct evidence to help us answer the question "Why Microcontrollers?", and has been a gratifying affirmation of our technology shift.

7. ACKNOWLEDGMENTS

Thanks to Max Mathews, Wendy Ju, Stefania Serafin, Chris Chafe, Gary Scavone, Fernando Lopez-Lezcano and the CCRMA and music department administrators. Thanks to all of the participants in the 2002 summer workshop who were good sports as our guinea pigs. Thanks to Chad Cosby, David Lowenfels, Gregor Hanuschak, David Lowenfels, Sara Shaughnessy, and Renee Goldschmid.

8. REFERENCES

[1] Atmel. `http://www.atmel.com/`, January 2003.

[2] Avr gcc compiler tool chain. `http://www.avrfreaks.net/`, January 2003.

[3] Linx technologies. `http://www.linxtechnologies.com/`, January 2003.

[4] Maxim 1270 12-bit adc. `http://www.maxim-ic.com/`, January 2003.

[5] Open sound control. `http://cnmat.cnmat.berkeley.edu/OSC/`, January 2003.

[6] Parallax inc. `http://www.parallax.com/`, January 2003.

[7] uisp - universal in system programmer. `http://savannah.gnu.org/projects/uisp/`, January 2003.

[8] M. Puckette. Pure data. `http://crca.ucsd.edu/~msp/software.html`, January 2003.

[9] P. Stang. Avrlib: C function library code for atmel avr processors. `http://www.procyonengineering.com/avr/avrlib/`, January 2003.

[10] P. Stang. Avrmini: Homepage of the diminutive atmel avr application/development board. `http://www.procyonengineering.com/avr/avrmini/`, January 2003.

[11] B. Verplank, C. Sapp, and M. Mathews. A course on controllers. In *NIME 2001*, March 2001.

[12] S. Wilson. Ccrma summer physical interaction design workshop website. `http://www-ccrma.stanford.edu/workshops/PIDI2002/`, December 2002.

[13] S. Wilson. Osc serial object. `http://www-ccrma.stanford.edu/~rswilson/OSCSerial`, August 2002.