MetaCricket: A designer's kit for making computational devices

by F. Martin
B. Mikhak
B. Silverman

We present the design of a construction kit, for building computational devices, called MetaCricket. MetaCricket consists of a set of hardware modules and the integrated software, which runs both on a development computer and within the MetaCricket hardware. MetaCricket provides a flexible interactive development environment for trying out new hardware and behaviors. The underlying architecture makes it easy for designers to expand the basic construction kit themselves with minimal engineering effort. Through a few examples, we show how designers, enabled by MetaCricket to be engineers of their own tools, are rethinking and transforming the very character of design principles in the digital age. MetaCricket was originally designed for use by children, but has been adopted by professional designers who are not professional engineers. These designers have found it incredibly liberating to directly implement their ideas without depending on engineers for assistance.

We are witnessing a revolution that will change the way we think, live, and play. By the end of this decade, computers will be imbedded in everyday objects all around us: our home appliances and furniture, our communication and transportation devices, and even our books and our clothing. The inevitability of ubiquitous computational resources has presented today's designers with both an opportunity and a challenge.

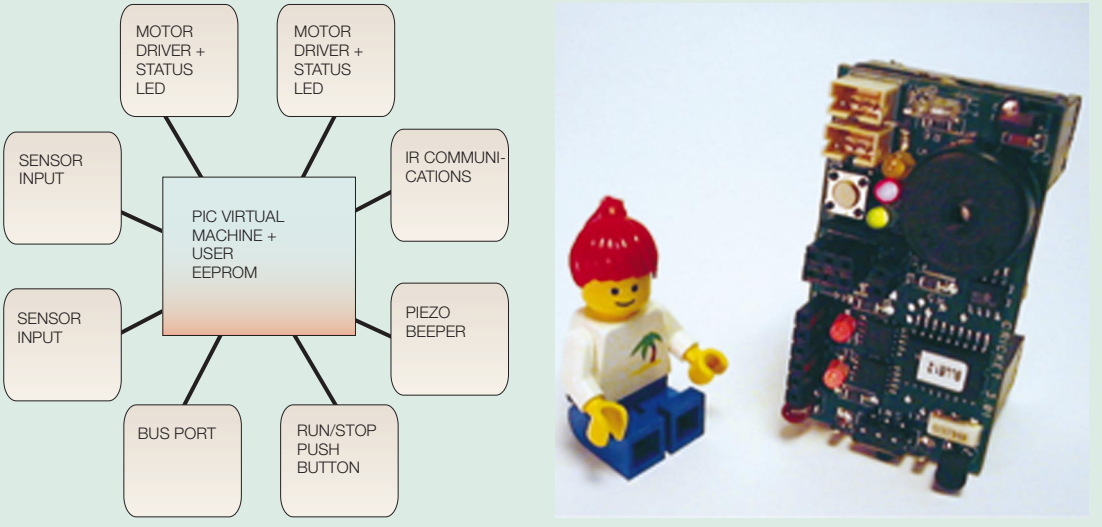
Many designers have seized this opportunity to produce more and more responsive and interactive artifacts and spaces. In their new designs, they pay close attention to the quality of the behaviors and modes of interaction as well as to the aesthetics of the static structures they create. Computation, while enriching the range of functions offered, makes it more challenging to achieve harmony between form and function. A new generation of technically savvy designers has embraced this challenge and is actively expanding and fundamentally rethinking traditional guidelines for good design.

This clearly is an exciting time to be a designer, but unfortunately, many talented designers, who are not accustomed to formulating their design concepts to engineers before iterating on them, are excluded from the excitement. Today, there are many software tools for graphic designers, animation artists, and digital-effect designers in general, but there have been few hardware systems that allowed designers to easily incorporate sensing, actuation, and programmable logic into their prototypes and designs.

Beyond the benefits of giving expressive power to individuals, companies and organizations need to support rapid prototyping practices. Schrage and Peters provide evidence for this in *Serious Play: How the World's Best Companies Simulate to Innovate*.[1] As noted on the book cover:

> Contrary to the popular assumption that innovative teams generate innovative prototypes, in fact innovative prototypes generate innovative teams.

Figure 1 Cricket block diagram and photograph. The Cricket is based on the PIC16C715 microprocessor with 4096 bytes of EEPROM for users' programs. It is powered by a 9-volt battery and includes output drivers for two motors, input drivers for two sensors, a piezo beeper, bidirectional infrared communications, a push button, and a peripheral expansion port.



> How innovators play with their models and simulations invariably matters far more than what they actually plan. In fact, [Schrage and Peters show] why innovative firms cannot seriously plan unless they seriously play.

Indeed, the MIT Media Laboratory where we work is an ideal representation of the sort of culture in which rapid, functional prototypes are taken seriously. These prototypes, which serve as objects of reflection, are critical in the evolution of most projects. The prototypes anchor discussions between lab sponsors and researchers, allowing projects to move from an academic level to their practical implementation. In other words, the prototype demonstrations make ideas concrete and push them forward.

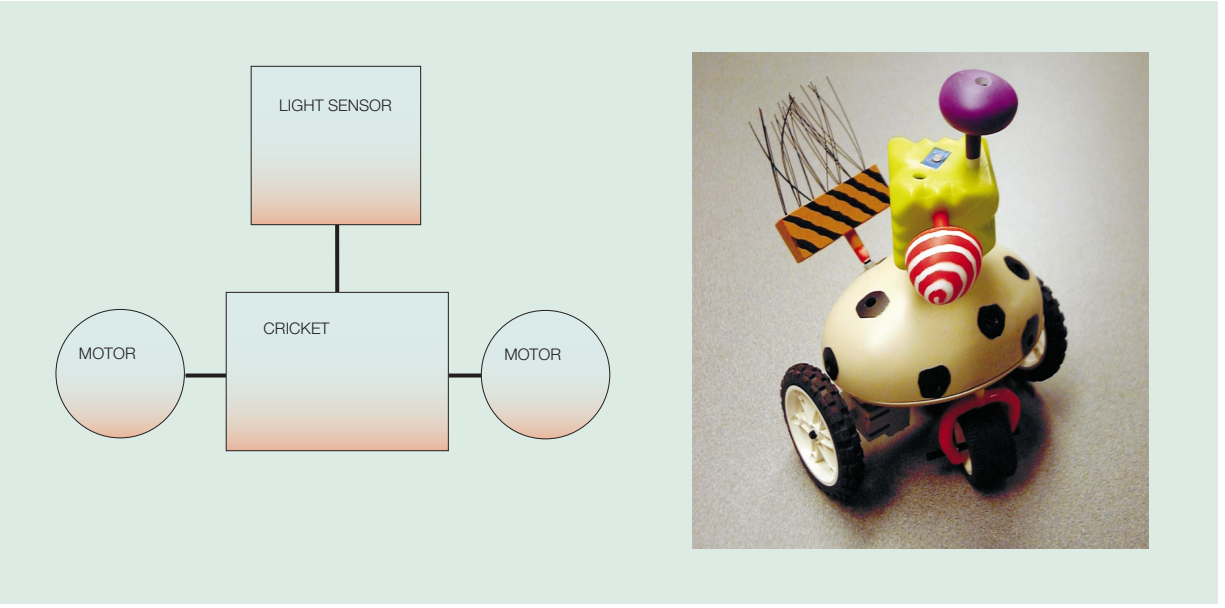
In this paper, we introduce *MetaCricket*, a hardware and software construction kit for building computational devices. MetaCricket has two crucial properties. First, it allows designers of all backgrounds, not just engineers, to create working prototypes of their ideas, ready for honest critique, analysis, and feedback. Second, the MetaCricket system is itself easily extensible. It is growing continuously, and is customizable by designers who have modest hardware and software backgrounds.

This paper describes MetaCricket through a series of examples of increasing complexity. The next section presents the Cricket processor at the heart of the MetaCricket system, along with a sample project. The third section presents the software architecture of MetaCricket. In the fourth section, the bus architecture—the basis of MetaCricket's expandability—is described in detail along with numerous examples of its use. In the fifth section, we use the MetaCricket architecture as a philosophical and practical foundation for the design of new devices. The discussion in the remainder of the paper includes our plans for future work and our concluding thoughts.

The MetaCricket core

In this section, we introduce the Cricket, the centerpiece of the MetaCricket system, and describe an introductory project that uses it.

Figure 2 Electronic dog block diagram and photograph. "Dr. Martin" is built using a Cricket that directly controls two motors and receives input from a light sensor.



The Cricket. The heart of MetaCricket is the Cricket, a tiny, programmable computer (about the size of a 9-volt battery) that can directly control motors and receive information from sensors (see Figure 1). 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.[2,3]

The Cricket is based on a Microchip PIC** microprocessor.[4] 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.

Crickets come with a custom software environment, known as *Cricket Logo*. Cricket Logo is a procedural language that includes global and local variables, procedure arguments and return values, control structures like repeat and loop, if and ifelse, and specialized primitive functions for interacting with motor and sensor hardware.

In addition, Cricket Logo is based on an iterative, interactive model of project development. It includes a "command center" window; instructions typed into this window are instantaneously compiled, downloaded to a Cricket, and executed, giving the system the flavor of an interpreted software environment such as LISP, BASIC, or FORTH. For the beginner, this interactive capability makes learning the system much easier, since trying out a new idea is as simple as keying it into the command center. For the expert, the command center encourages an incremental approach to project building, allowing easy interaction with the program under development.

Dr. Martin: An electronic dog. With just the core Cricket device, common DC motors, and simple sensors, a wide collection of computational devices can be easily prototyped. Consider "Dr. Martin," an electronic dog shown in Figure 2, created as a learning exercise by a nontechnical professional toy designer over a one-week period. Its body is constructed largely of parts from the ZoLO** toy construction

kit, but its control electronics are built from the Cricket kit. The dog has two LEGO motors for movement, a Cricket in its belly, and a light sensor in its nose. Next to the photograph of the model, the figure shows the functional block diagram of the dog.

Dr. Martin was designed with a simple "follow the owner" behavior, implemented in Cricket Logo. The software checks the light sensor and moves the dog forward if the light level is high enough. Waving a hand in front of the dog will cause room lighting to be reflected into the sensor, and the dog then moves forward. So by keeping your hand in front of the dog, you can get him to chase you.

This program just takes a few lines of code:

```
to follow
  loop [                ; begin infinite loop
    if sensora > 50     ; check light level, if high then
      [ab, onfor 20]    ; motors A and B on for 2 sec
    wait 5              ; wait 1/2 sec try again
  ]                     ; end loop
end
```

The program, named follow, sets up an infinite loop. Inside the loop, the light level is checked. If it is greater than 50, then the motors are turned on for two seconds. After a half-second delay, the loop recycles.

The threshold value of 50 would not work for all room lighting conditions. With a more sophisticated approach, the model could behave properly in the more general case. Nevertheless, the example makes the point of how simple it is to prototype novel computational devices using the Cricket.

This section has introduced the core Cricket, including an overview of its hardware, software, and an application example. The next section examines the software design of the Cricket system, then discusses the Cricket peripheral expansion bus and ways in which the whole Cricket architecture can be reconfigured for custom applications.

Cricket software

The MetaCricket software system is based on a virtual machine, written in PIC assembly language and running on the Cricket, and a compiler for the virtual machine running on a desktop development computer, with two implementations (one in Logo and one in Java** code). This approach yields several important benefits:

- The object code resulting from compiling the user's program is quite small, and easily fits on an inexpensive serial memory chip (included in the core Cricket design).
- The virtual machine has a simple, stack-based architecture, allowing both its implementation (in PIC assembler language) and its compilers to be written with a minimal amount of code.
- It is straightforward to implement an interpreter-like interface, where user commands are transparently compiled, downloaded, and executed.

This section describes the implementation in detail. We will begin with the virtual machine implemented on the Cricket, and then discuss the desktop compiler and communications protocols.

The Cricket virtual machine. The Cricket virtual machine is burned into the PIC microprocessor's internal ROM (read-only memory) and is implemented in assembler language. The virtual machine uses the microprocessor's internal RAM (random-access memory) for a program stack. The user's code resides in a serial EEPROM (electrically erasable programmable read-only memory) that is permanently connected to the microprocessor. Built-in infrared communications routines include a protocol for reading and writing to this external EEPROM, and for asking the virtual machine to begin execution of byte codes already loaded into the EEPROM.

Early versions ran first on the PIC16C84 model, then on the PIC16F84. Our present version runs on the PIC16C715 and similar processors. This is a processor running with a 1 megahertz (MHz) instruction clock with just 128 bytes of internal RAM and 2048 words of internal ROM.[5] Table 1 illustrates features of the virtual machine across PIC versions. There have been three major revisions of the Cricket virtual machine since the project began in early 1996. The versions are known by their "dot color" since we applied colored dots to mark the programmed microprocessors. Access to EEPROM containing the user code provides the main bottleneck with regard to the speed of the virtual machine interpreter. Because the memory access is slow, the interpreter is limited to a speed of about 300 virtual machine instructions per second. This would seem to be very slow, but in practice, this limitation is all but inconsequential. If a peripheral device has high-speed requirements, it is handled with an additional dedicated

Table 1 Versions of the Cricket virtual machine

Virtual Machine Version	Processor Model	Capabilities
"Green dot"	PIC16C84	8-bit number system 8-bit program addressing with 2 kilobyte EEPROM (256 bytes program, 1792 bytes data) 40 kilohertz-based IR communications
"Red dot"	PIC16F84	8-bit number system 16-bit program addressing with 2 kilobyte EEPROM (1024 bytes program, 1024 bytes data) 40 kilohertz-based IR communications
"Blue dot"	PIC16C715	16-bit number system 16-bit program addressing with 4 kilobyte EEPROM (2048 bytes program, 2048 bytes data) IrDA-based IR communications Bus communications capability

PIC chip (as described later). This leaves the Cricket responsible for only the basic control flow or state machine aspect of a design, for which an update rate of about 100 hertz (Hz) is adequate.

User-level primitive functions compile to one, two, or three bytes of object code for the Cricket virtual machine. A stack holds expression computation results and arguments for procedures. In our byte code language, the expression "1 + 1" would compile to number 1, number 1, plus. During execution, the first number 1 would cause the value 1 to be pushed onto the stack. After the second number 1, another 1 would be pushed onto the stack. The plus operator would then pull the two 1s off the stack and push 2, the result of the operation.

User-defined procedures supplement the primitives provided by the language; the procedures can accept an arbitrary number of inputs and optionally produce an output. The same stack used for calculation of numeric results is used to hold return addresses of procedure calls. Because of the small amount of RAM provided in the PICs we use, nested and recursive procedure calls are limited to about 16 levels deep.

A 16-bit number system provides signed integer arithmetic. Standard arithmetic operators, comparison operators, and Boolean operators are provided. There is also a pseudorandom number generation function.

There are global variables for holding program state. Procedure inputs act as local variables. Additionally, 2048 bytes of the 4096-byte physical memory is set aside for extended data storage; an arbitrary number of linear arrays may be declared in this area. This array memory, along with the program memory, is persistent through power cycles.

Several control structures are provided. If-then and if-then-else tests, loops (repeat for a specified number of times, or loop indefinitely), and a wait-until-Boolean-expression-becomes-true structure are available. Additionally, there is a timed wait instruction (e.g., wait for one second).

The Cricket virtual machine has two process threads: a foreground process and a background daemon. In most Cricket programs, the foreground thread handles all the work, but for some tasks, the background daemon is valuable. For example, the background daemon can be used to instigate a periodic activity, or take action when some event occurs.

There are hardware-specific primitives for interacting with on-board Cricket hardware. Motor commands set state (on or off), direction, and power levels for each of the two integrated motor drivers. Analog sensor primitives (sensora and sensorb) return a value (0 to 255) for each of the two voltage inputs. These inputs also may be interpreted as digital values using the switcha and switchb primitives. There is a pair of primitive functions for generating tones on the piezo beeper: beep and note, the latter taking pitch and duration arguments.

Several features allow programs to keep track of real time during their execution. Simple timed commands

Table 2 Cricket Logo feature overview

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

provide a fixed delay with a resolution of 100 milliseconds (e.g., onfor 35 turns motors on for 3.5 seconds). Additionally, there is a background millisecond timer that is updated every four milliseconds; this timer counts from 0 to 32.768 seconds before "wrapping around" back to 0.

Table 2 summarizes the language features of the Cricket virtual machine. On a historical note, the Cricket's software system grew from the approach taken by Sargent and Martin in developing Interactive C for the 6.270 LEGO Robot Design Competition,[6] which itself grew from earlier work done by Martin and Silverman in developing the first MIT Programmable Brick.[2]

The Cricket Logo compiler. Users write programs for the Cricket in *Cricket Logo*, a dialect of Logo specialized for the Cricket virtual machine. Essentially, there is a one-to-one mapping between statements in Cricket Logo and primitive functions built into the virtual machine. This makes the implementation of the compiler far simpler than typical compilers.

Indeed, the reference compiler has only four pages of code. It is itself implemented in a version of Logo. We have also created a version of the compiler written in the Java language; it has a similar structure to the Logo implementation.

The compiler includes an interactive mode—a text window where user expressions are compiled, downloaded, and executed in one step when the user presses the return key. A portion of the Cricket's memory is set aside for these dynamic programs. To the user, this gives the impression of directly "talking to" the Cricket. After downloading programs, users can call their own procedures, providing various arguments, etc., all from the interaction window.

Figure 3 illustrates the user interface for the Cricket Logo compiler. The software is designed for use by novices and the interface is deliberately kept minimal. On the right is the user procedures window. In the lower left is the interaction window, known as the "command center." In the upper left are text fields (numbered 1 and 2) for single Logo statements. When the user's code is downloaded, the first of these two statements may be triggered by pressing a push button located on the Cricket itself. This allows downloaded programs to be started independently of the development computer. An infrared (IR) instruction may be also issued, causing the Cricket to execute either of the two numbered Logo statements.

The Cricket Logo environment does not include a specific debugger. In practice, we have found that a debugger is not necessary because of the interactive and incremental style of project development that occurs when using the Cricket. To make a change in the code, then download, and run is so rapid that work becomes more like a conversation with the artifact than an experimental process. Additionally, one can transmit numeric values back to the desktop computer for display, or add a display device to a project to show internal state variables.

Infrared communications. The Cricket includes a protocol for interaction via its bidirectional infrared capability. The Cricket implements a custom byte-oriented communications scheme based on IrDA (Infrared Data Association) hardware. The byte-level transport operates at an effective data rate of 50 thousand baud; this is actually faster than the Cricket can access its EEPROM.

The infrared protocol includes the following capabilities:

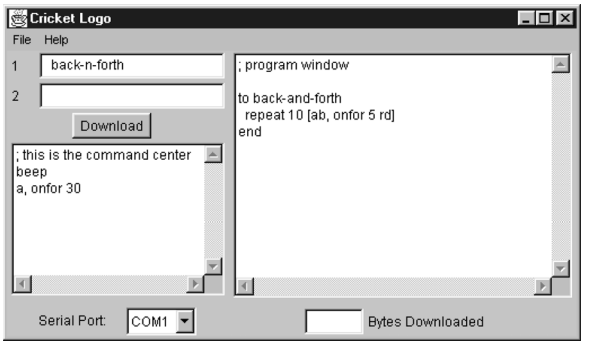
- Check that a Cricket is present and ready for other commands.
- Write a byte to the Cricket's EEPROM.
- Read a byte from the Cricket's memory.
- Begin program execution from a particular memory address.

In sum, the core Cricket design is simple yet powerful; its software is basic yet expressive. In the next section, we show how the Cricket can be the heart of a larger system.

The bus architecture

Thanks to its ease of use and small size, the original Cricket processor became very popular. Designers soon envisioned more and more elaborate applications that required new functionalities for the crickets. To address the limitations of the first Cricket design (two motors and two sensors), we extended the design along two parallel directions.

Figure 3 Cricket Logo interface

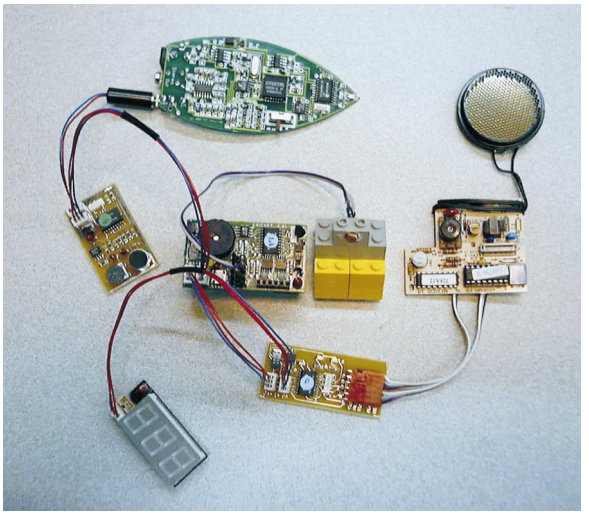


On the one hand, since crickets were simple and easy to revise, we created a plurality of Cricket designs. We designed customized crickets for each new class of projects; for example:

- The "display Cricket," designed for applications that needed a richer display and more sensor inputs, had a bank of eight bicolor (red/green) LEDs (light-emitting diodes) that could be lit under user program control, three sensor inputs, but no motor outputs.
- The "MIDI (Musical Instrument Digital Interface) Cricket," designed for musical applications, had a versatile waveform synthesizer chip, more sensor inputs, but no motor outputs.
- The "science Cricket," designed to help children build their own scientific instruments, provided true analog-to-digital converters on the sensor inputs (allowing the use of a greater variety of sensor devices), along with support for 16-bit numbers in the Cricket software. (The science Cricket employed a more powerful version of the Microchip PIC controller.)

On the other hand, we developed a different way to support multiple devices—the Cricket bus system. As we were designing devices to interface with the Crickets, we realized that often a custom circuit would be required to connect a particular device to a Cricket. Up to this point, we had been designing entirely new Crickets with different capabilities. For ex-

Figure 4 Cricket with bus devices. At center is the "blue dot" version of the MIT Cricket with a simple light sensor to its immediate right. Arranged around, from top going clockwise, bus devices: MidiBoat music synthesizer, Polaroid ultrasonic distance sensor, 3-digit numeric LED display, and sound sensor.



ample, to get an LED display, we created the display Cricket; to get music, we created the MIDI Cricket.

We came to realize that instead we could bundle the specific circuitry required to make a new device work with that device itself, in a sort of object-oriented hardware strategy. Then, a simple communications protocol could let the new devices talk to an existing Cricket. Thus was born the idea for the Cricket bus.

The Cricket bus allows an arbitrary number of devices to be daisy-chained off of the bus port. Each bus device includes two bus port connectors to allow a series of bus devices to be connected in a string. For flexibility in wiring configurations, two bus ports are provided on the Cricket (though electrically they are the same signal).

Our first Cricket bus device was an optical distance sensor that employed a special part manufactured by the Sharp Microelectronics Group. This component used a specific, unusual communication method to interface with an external circuit. By conceiving the Sharp distance sensor as a bus device, we were able to bundle the special hardware and software required to talk to the device with the device itself, yielding a common protocol to talk between the Cricket and the bus device. Over time, we have come to develop a large collection of bus devices, all of which can communicate with a standard Cricket. Figure 4 shows a sampling of these bus devices, including a music synthesizer called the MidiBoat, an ultrasonic distance sensor, a three-digit LED numeric display, and a sound sensor.

The bus system also allowed us to converge on a single Cricket design. Our "classic Cricket" (with two built-in motor drivers and two resistive sensor ports) merged with the "science Cricket" (true analog sensor inputs and 16-bit number support) to become, with the addition of the bus port, our standard design. The need for different Cricket versions was drastically reduced, since nearly any device can be interfaced to this standard Cricket using the bus system.

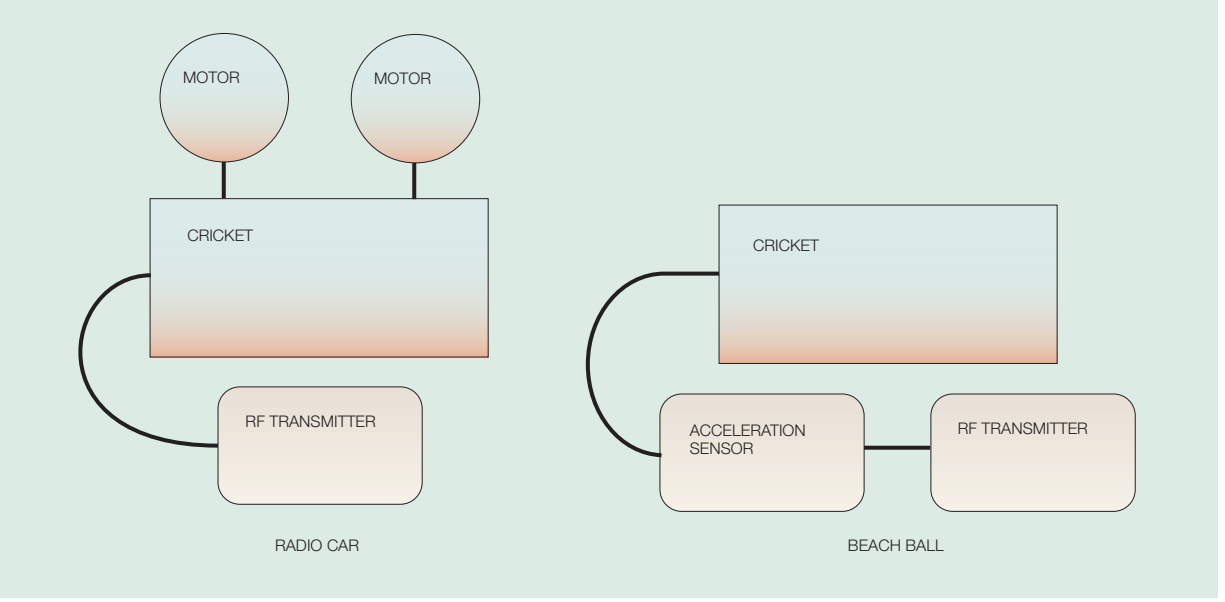
We next describe a variety of projects, built by ourselves and our colleagues, that demonstrate the range and expressive power of the MetaCricket system.

Beach ball remote car. Much of the material we have developed for MetaCricket has been used by teachers and students in classroom settings. In one particular case, a high school student who had an extended relationship with our work designed a remote-control toy: a motorized LEGO vehicle with (1) a Cricket for controlling the motors and the bus devices, and (2) a radio frequency (RF) receiver bus device. The car ran a simple program that guided its movements based on received RF commands:

```
to car
  loop [                          ; begin infinite loop
    waituntil [newrf?]            ; wait until new RF
                                    command received
      if (rf = 1) [fd onfor 2]    ; if 1, drive forward for
                                              0.2 sec
      if (rf = 2) [bk onfor 2]    ; if 2, drive backward
      if (rf = 3) [rt onfor 2]    ; if 3, drive right
      if (rf = 4) [lt onfor 2]    ; if 4, drive left
  ]                               ; end loop
end
```

Instead of the traditional steering wheel controller, this project used a Nerf** ball with MetaCricket components embedded inside. The ball contained (1) a Cricket, (2) an RF transmitter bus device, and (3) a two-axis accelerometer bus device used as a tilt sensor.

Figure 5 Beach ball remote car block diagram



The Nerf ball controller ran a program to generate the RF command bytes (1 through 4) depending on the data from the acceleration sensor, which indicated how the ball was tilted. When the student held the ball in two hands and rotated it in a "rolling forward" motion the vehicle moved forward, and when the student rotated it in a "rolling sideways" motion, the vehicle turned correspondingly.

Figure 5 shows the block diagram of the resultant system. This simple example illustrates how a novel physical interface to a conventional toy is easily prototyped.

Communicating musical toys. Over the past couple of years, communicating musical toys have been prototyped by various designers around the Media Lab. All of these toys have the following components:

- Crickets
- The MidiBoat bus device, a full MIDI synthesizer on a small board[7] that accepts MIDI music commands over the Cricket bus
- Push buttons, pressure sensors, and other unusual sensor inputs

Because Crickets have built-in infrared communication capability, they are especially suited for projects in which multiple computational devices must interact with one another. Most of the musical toy projects have this quality.

The MidiBoat bus device is another important component. This board was originally designed for a specific Media Lab project that was not Cricket-based. Later the firmware on the MidiBoat device was reprogrammed to accept Cricket bus commands, and the board has seen many new, unintended applications. Several projects illustrate the virtuosity of MetaCricket for these musical applications.

Play-Doh synthesizer. This project is inspired by the "music shapers" approach described by Machover et al.[8] In this project, a glob of Play-Doh** is connected to a personal computer (using a resistive sensor interface), and algorithms on the computer generate music based on the changing resistance of the Play-Doh as it is manipulated.

In our implementation, the Play-Doh is connected directly to a Cricket, which drives a MidiBoat synthesizer (Figure 6). The sensor is made up of two pairs of electrodes, each plugged into one of the two Cricket sensor ports, and some Play-Doh (or cookie or bread dough) for establishing electrical contact between the electrodes. As the Play-Doh is worked

Figure 6 Play-Doh instrument block diagram. The Play-Doh sensor is literally just two wires stuck into a glob of Play-Doh. Salts in the Play-Doh form a current path with a varying resistance as the dough is reshaped. This resistive input is measured by the Cricket, and a simple control program generates music based on it.

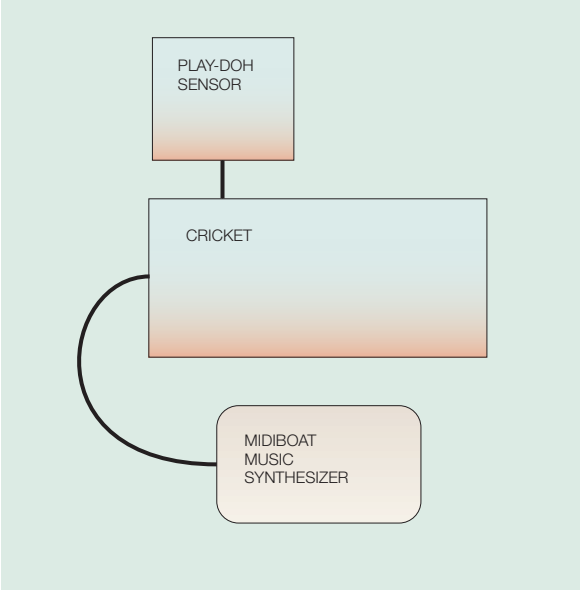
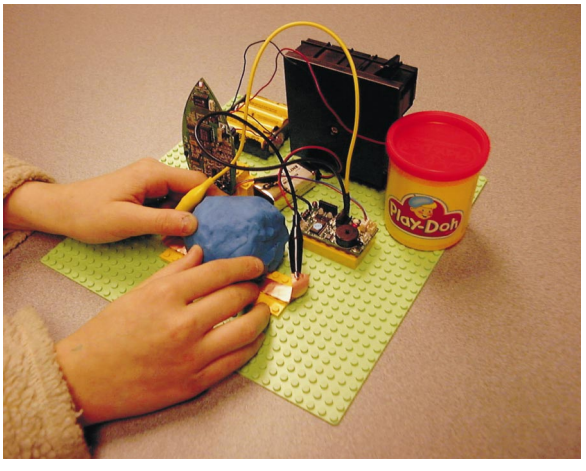


Figure 7 Play-Doh instrument in action



over and reshaped, the sensor readings change. These readings are mapped to high-level musical concepts like melody contour and tempo.

Figure 7 shows the working Play-Doh instrument. One can reprogram the Cricket to change not only the mappings but also the instruments' voicing, volume, and any other qualities. Other sensor bus devices can be used to design any number of interfaces for controlling these parameters.

Musical sensory puzzle. The musical sensory puzzle consists of a number of blocks, each containing a section of a musical piece (Mozart's *A Little Night Music*). The user can listen to each section separately, by pressing a button on one of the blocks, or arrange the blocks in a linear chain and listen to that arrangement played sequentially. The user can rearrange the blocks to discover how new arrangements sound and maybe rediscover the entire piece as arranged by the composer.[9]

To accomplish this, each block must communicate with its adjacent neighbors by IR. Each Cricket includes infrared communications facing out one end; in order to "talk" out the other end, an infrared communications bus device is used. Thus, the musical sensory puzzle is built from a Cricket, an additional communications device, and the MidiBoat synthesizer (Figure 8).

SqueezeMan. The "SqueezeMan" is a hand-held musical device that maps squeezing gestures to high-level musical concepts like melody contour and tempo. Together, these devices offer a range of novel musical experiences that take advantage of their ability to interact and interdependently manipulate each other's musical outputs. The SqueezeMan devices extend and transform the ways in which we interact not only with our traditional musical instruments but also with people around us. They were designed to draw users to a deeper, more meaningful active musical participation.[10]

Each SqueezeMan device consists of a semi-transparent plastic container (a child's drinking cup) as a casing (Figure 9). The electronics and batteries are embedded in the cup while the input and output devices are mounted on its top. Two plastic "eyes" are glued on the body of the cup to indicate the location of the infrared components of the Cricket embedded in the device. This device transmits and receives infrared signals through the semi-transparent plastic.

Two force-resistive pressure sensors are embedded inside a cluster of four squeezable rubberized star-shaped balls. The sensors generate continuous volt-

Figure 8 Musical sensory puzzle block diagram

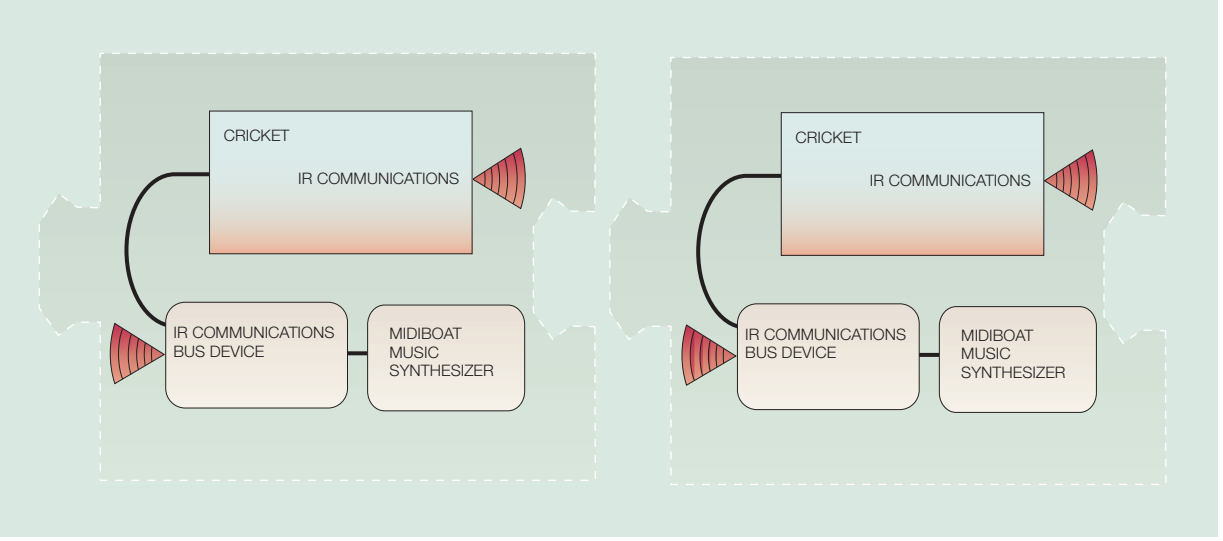
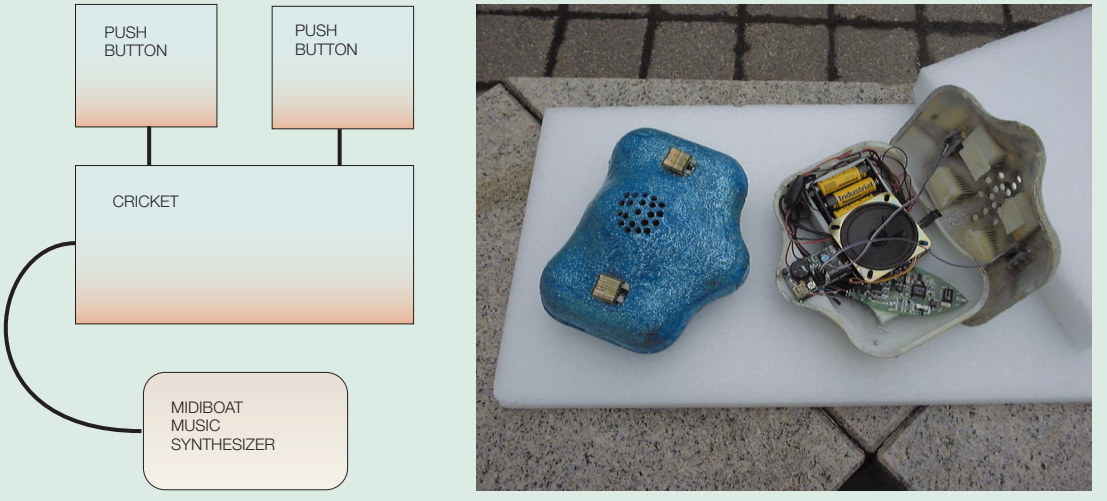


Figure 9 SqueezeMan block diagram and photograph

ages in response to the pressure levels exerted on each of the two independent pressure axes in the cluster. Squeezing was chosen as an intuitive and expressive gesture for high-level, continuous control. The ergonomic design allows for two-handed manipulation, using each palm to press against one pair of rubber stars. More experienced users can operate the device with one (playing) hand (a thumb-forefinger axis and a thumb-middle finger axis). This allows a performer to hold the device in the other

Figure 10 Musical fireflies block diagram and photograph



(nonplaying) hand and point it toward other performers' devices in multiplayer mode.

The pressure sensors are wired to the Cricket's built-in analog input ports. The Cricket runs a program that interprets the pressure data and maps it to musical messages sent to the MidiBoat device.

The MidiBoat device's audio-out jack is connected to a headphone jack, which is mounted on the device's top. The use of headphones (instead of speakers) imitates the Sony Walkman** solution to the challenge of providing personal local high-quality sound. Like the Walkman, this solution avoids the problem of speaker quality, weight, and power consumption common to hand-held speaker-based devices.

Musical fireflies. Musical fireflies (see Figure 10) are digital toys that introduce mathematical concepts in music, such as beat and rhythm, without requiring users to have any prior knowledge of music theory or instruction.[11] Each firefly has a Cricket, a MidiBoat, and two buttons—one for entering accented notes, the other for entering nonaccented notes. When the firefly is first turned on, it waits for the user to enter a rhythm pattern. The left button records an accented beat; the right button records a nonaccented beat. After two seconds of inactivity, the firefly plays this pattern. During playback, players can input a second layer of accented and nonaccented notes in real time. Each tap on a button plays a beat aloud and records its quantified position so that the beat becomes part of the rhythm loop. Pressing both buttons simultaneously at any point stops the playback and allows the player to enter a different pattern.

Musical fireflies interact through their Crickets' infrared communication capabilities. When two fireflies "see" each other, they automatically synchronize their rhythm patterns. While two fireflies are synchronized, users can initiate a "timbre deal" in which instrument sounds are traded between the toys. Pressing either the left or right button trades the instruments used to play the accented or nonaccented beats, respectively. Each firefly now plays its original pattern but with new timbres that bring a new perspective to the rhythmical interaction. The interaction between two musical fireflies makes them both richer; they both contain four sounds after the interaction.

In the current design of the musical fireflies, directionality of the IR communication limits simultaneous interaction to three devices. Using the RF bus de-

vices one can remove this limitation and make new musical performances and games possible by allowing a room full of musical fireflies to interact with each other.

Dance craze buggies. The dance craze buggies are a demonstration prototype of an Internet-generation children's toy, based on the idea of "tradable bits."[12] Tradable-bits toys let children trade bits (including pieces of music, animations, and digital creatures) instead of atoms (such as the traditional baseball cards, POG disks,[13] and Beanie Babies**). Trading digital (rather than physical) objects has distinct advantages: children can more easily author, copy, modify, and trace them.

The dance craze buggies themselves (Figure 11) are a set of modified radio-controlled cars, which can perform dances when they "see" each other. Further, the buggies can trade dances with each other and also upload information about how the dances are spreading to the Internet. Each buggy has its own two-way pager (using Motorola CreataLink** technology) that allows it to communicate wirelessly to a server on the Internet.

In the prototype implementation, children can play with their buggies, get them to teach each other dances, and view on-screen visualizations of how the dances have spread from buggy to buggy. The goal of the project is to allow children to become reflective about the spread of ideas through a culture; in direct play with the buggies, they can experience first hand the exchange of dances, and then later, with visualizations, explore the results.

The dance craze buggies' hardware was prototyped within a week by the one of the authors. Figure 12 illustrates the block diagram of the buggy. Each car was a radio-controlled car modified to have:

- A Cricket that controlled the motors and took care of the communication with the other buggies through Cricket IR
- Two three-digit LED display bus devices programmed to display the abbreviated names of the dances the buggy knows
- A knob (a potentiometer plugged into one Cricket sensor port) that allows scrolling through the dances, and a button plugged into the other Cricket sensor port that allows a dance to be selected
- A two-way pager (Motorola CreataLink), modified to act as a bus device
- The MidiBoat music synthesizer

Figure 11 Dance craze buggies photograph

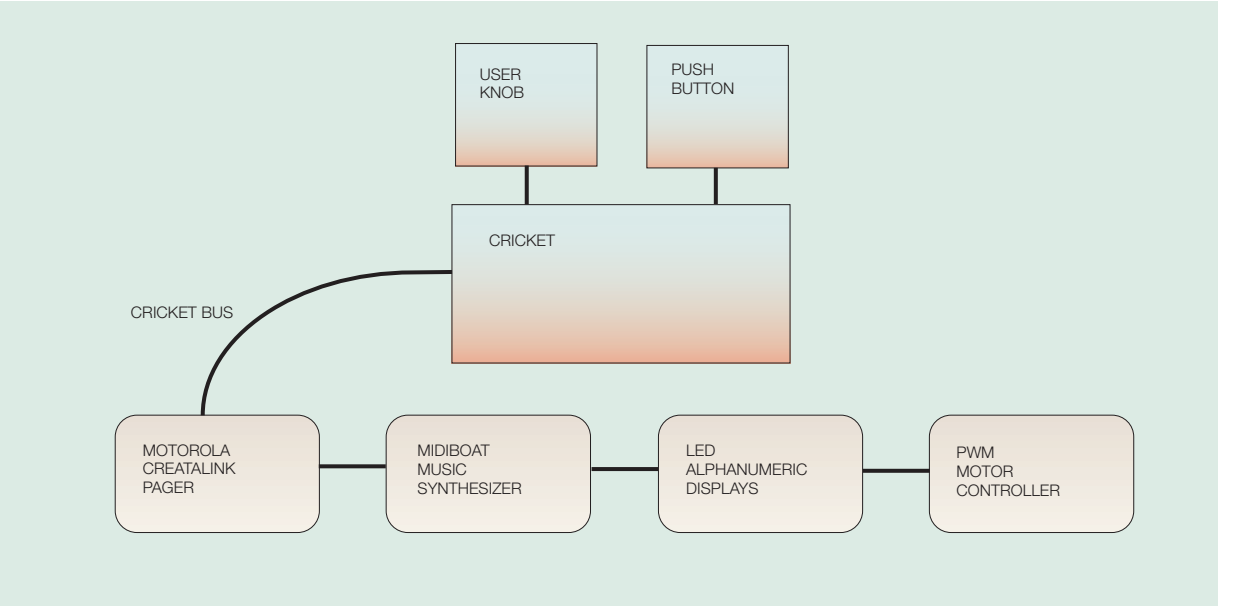
Anatomy of a bus device. The two motivations for the bus device system were to expand the Cricket beyond its built-in two-motor/two-sensor limit, and to bundle hardware specific to a given sensor or actuator with that device itself. Beyond serving that goal, the bus system has had additional benefits. A wide variety of hardware devices has been interfaced to the Cricket, including devices that were originally designed for other purposes (such as the MidiBoat synthesizer).

As this collection of devices has proliferated, propitious combinations have given creative freedom to designers using MetaCricket and made complex groups of devices easy to interface together (as in the dance craze buggy).

We want to encourage this process by demonstrating the ease with which a new bus device can be constructed. For tutorial purposes, we will present the design of a specific bus device for measuring resistance. Despite the fact that the Cricket device includes two built-in ports for measuring voltage, the resistance sensor is useful as a bus device for applications where more than two sensors are required.

Circuit details. Figure 13 shows the schematic for the resistance sensor bus device. It uses the PIC16F84 processor, with electrically erasable program memory that can be reprogrammed in-circuit, making it ideal for experimental purposes. This processor does not include an analog-to-digital converter, so the circuit employs the old trick of charging a capacitor

Figure 12 Dance craze buggy block diagram. In the dance craze buggy design, a Cricket talks to an adapted Motorola CreataLink two-way pager, a MidiBoat music synthesizer, LED alphanumeric displays, and a pulse-width modulated (PWM) motor controller.



through the unknown resistance and timing the charge cycle.

Aside from this circuit, and the connections required to support the processor, only one pin must be dedicated for the circuit to become a bus device. In the schematic, the bus communications line is Pin 6.

Low-level bus communications. The Cricket bus uses byte-oriented data transfer, with an additional bit included to distinguish between commands and data. The communications protocol follows a strict master-slave method: the Cricket device is the single master, and all of the bus devices are slaves. When the Cricket talks to the bus, it issues a command for a single bus device. That bus device may either silently take action, or it may respond by sending its "answer" back on the bus.

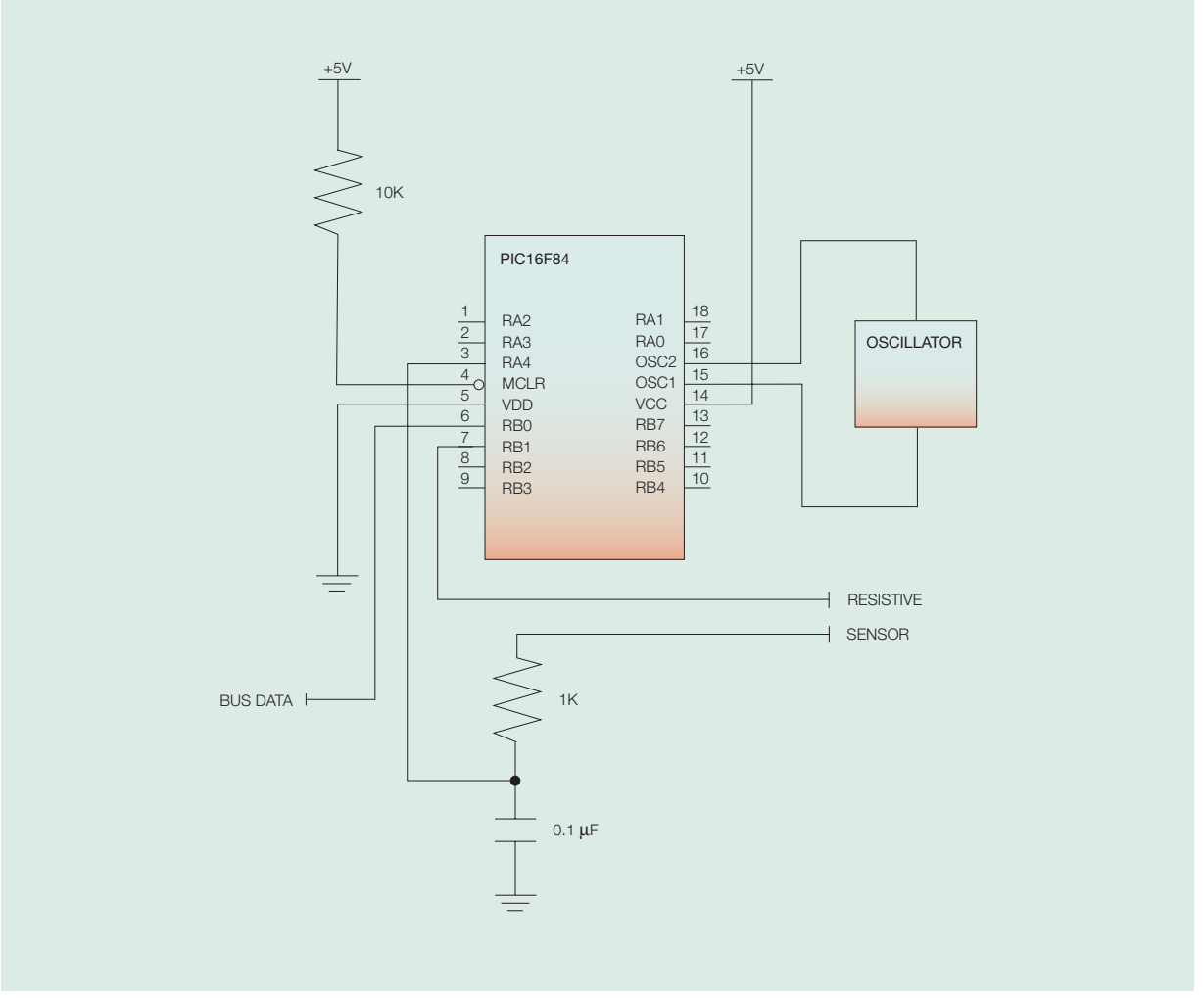
Figure 14 illustrates the low-level signal format. Each transmission consists of a 100 microsecond "prestart" synchronization pulse, which is followed by a start bit, 8 data bits (least significant bit first), a 9th bit to indicate command or data, and a stop bit.

The (relatively) long synchronization pulse allows all receiving devices to get ready to receive the bus data. This may mean either processing an interrupt (if the bus signal is connected to an interrupt pin) or finishing another task before resuming polling of the bus pin. In any event, all receiving devices are expected to dedicate their full attention to the bus signal before the synchronization period ends.

At the rising edge of the start bit, receivers synchronize their internal timing to allow the remainder of the communication to be performed at the high speed of 10 microseconds per bit. Then the eight data bits are transmitted, followed by the command/data bit and the stop bit.

Device communications protocol. The command/data bit of the low-level bus word is used to indicate whether the byte being sent is a command byte or a data byte. Command bytes may be sent only from the Cricket to bus peripherals, whereas data bytes can be the reply of a bus peripheral to the Cricket, or the additional parameters of a full bus command. Each bus peripheral has a unique 8-bit identifier, so any given command must address not more than one device on the bus. By tagging replies and additional arguments as data, other peripherals do not inadvertently interpret this additional bus traffic as command bytes.

Figure 13 Schematic of resistive bus device



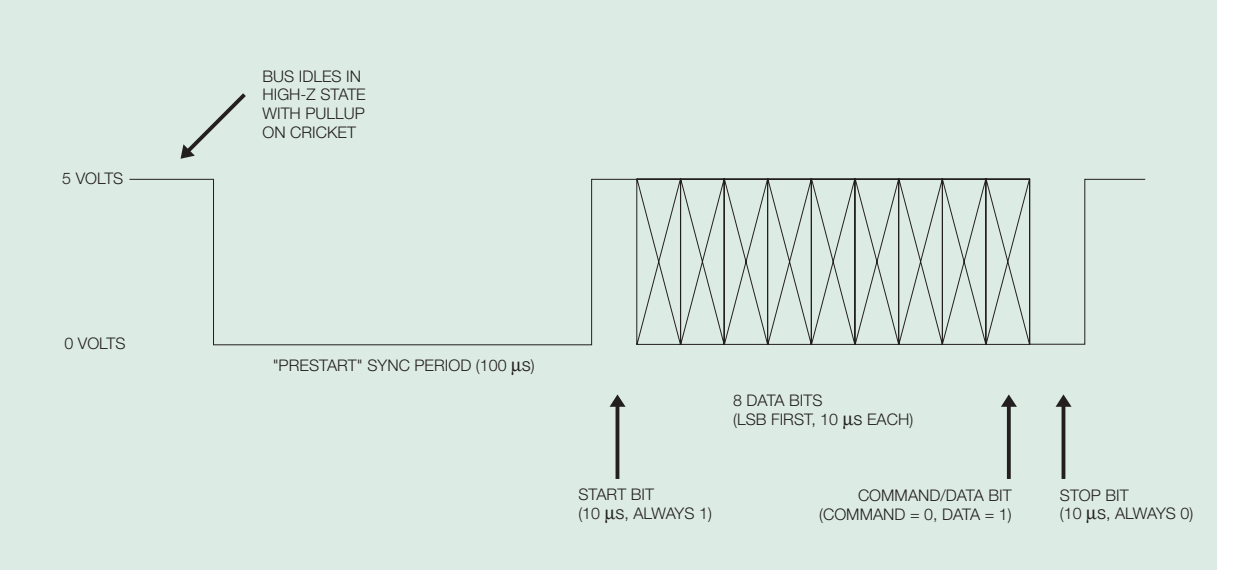
For instance, consider the following method by which the Cricket addresses the acceleration sensor (as used in the beach ball remote car project):

- Cricket sends "190" hexadecimal. The high-order bit marks the word as a command byte; the value "90" is the unique identifier for the acceleration sensor.
- Cricket sends "1." This number is retrieved by the acceleration sensor and interpreted as a request for the value of the first of its two axes.
- The acceleration sensor sends its reply (marked as a data byte), which is retrieved by the Cricket.

Communications subroutines. We have written short assembly language routines to transmit and receive in the Cricket bus protocol on three different processors: the Microchip PIC, the Motorola 68HC11, and the Hitachi H8**. The protocol is successful because it was designed to be easily implementable on any small processor without the need for specialized hardware.

Integration with Cricket Logo. We have implemented a simple protocol whereby many bus devices can be interrogated with a single byte command. More complex devices use the first byte of the bus command

Figure 14 Bus communications signal drawing



as an escape into an idiosyncratic protocol specific to the device itself.

For each kind of device (e.g., reflectance sensor, LED display), four unique identifiers are available—tagged by color (red, blue, yellow, or white). Cricket users refer to the device by color when creating Cricket programs (e.g., "get the value of the red reflectance sensor"; "display the value of the red reflectance sensor on the blue LED display"). This color identification system supplements the port-based identification method (e.g., Motor A, Sensor B) used by the Cricket's on-board peripherals.

More complex bus peripherals share the color naming system, but may have multibyte sequences for being addressed. For instance, the shaft encoder bus device uses a two-byte command sequence, where the first byte indicates which of the four color-named shaft encoders is to be addressed and the second byte indicates the operation to be performed: return position count, return current velocity, or clear position count.

The last step is to add a library to the Cricket Logo software in order to define higher-level primitive functions for accessing the bus device. This library routine may be written in Logo and can be preloaded onto a Cricket before repeated use. For example, the following subroutine displays a number on the red four-digit LED display:

```
to display :n
  bsend $170              ; $100 sets command bit,
                          ; $70 is device ID
  bsend high-byte :n
  bsend low-byte :n
end
```

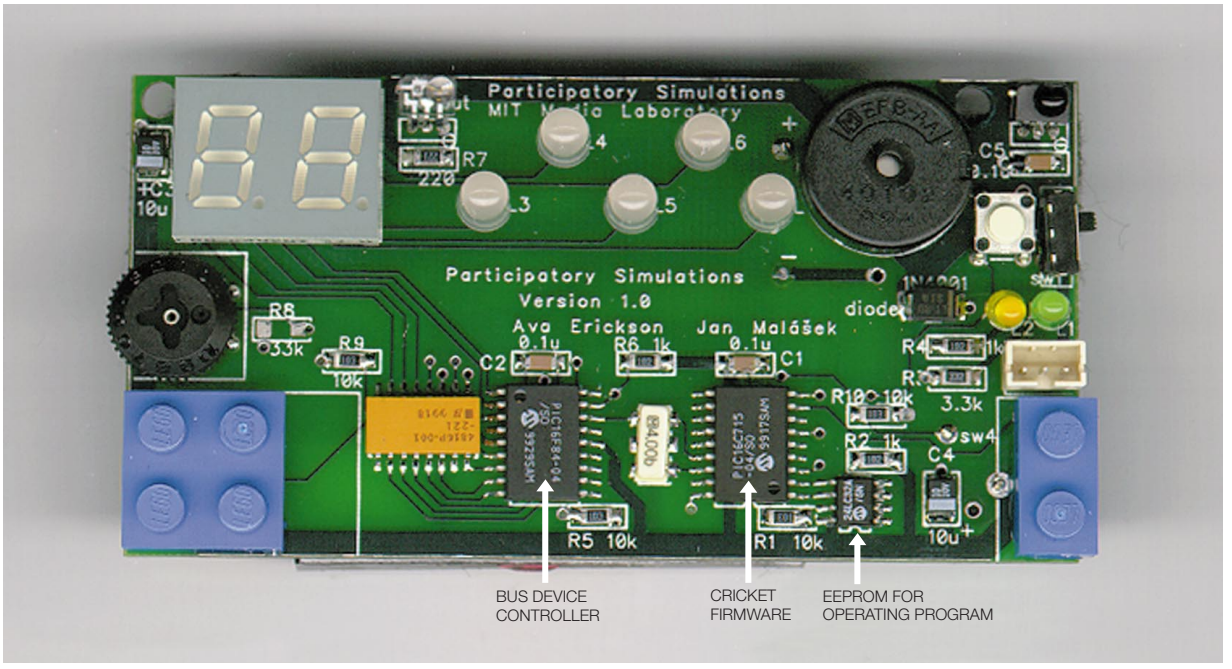
MetaCricket architecture as a building block

MetaCricket is more than the collection of bus devices along with the core Cricket device. It is an approach to building computational devices. Rather than attempt to integrate a master control program and drivers for various hardware devices into a single chip, MetaCricket proposes the use of a PIC processor, with the Cricket firmware, talking to assorted bus devices integrated into a single circuit design.

In other words, one can take an application built from the Cricket and several bus devices, and create a new design with this set of components laid out on a single board.

This section presents two projects that have been developed using this approach. In the first, the Cricket

Figure 15 Participatory simulation thinking tag



and several particular bus devices are integrated into the "participatory simulation thinking tag." In the second, the Cricket is embedded inside a 6×2 LEGO brick stack, with bus connections running up and down allowing these bricks to communicate, creating the "Tangible Program Brick." In describing these projects, we illustrate the effectiveness of using the components of MetaCricket as building blocks in integrated designs.

Thinking tags. In our early work with "thinking tags," we created electronic name tags that allowed users to learn about each other's preferences in social settings.[14] In an extension of this work, Vanessa Colella is developing *participatory simulations*. Students wear small communicating computers and become actors in life-sized, computationally supported simulations. For instance, students can participate in a simulation of a virus spreading through a community. Participants then draw on their own experiences as they work together to describe the behavior of the whole system.[15]

The first participatory simulation thinking tag was built using conventional design techniques. The Cricket operating system was extended to include drivers for a two-digit numeric LED and five bicolor LEDs. When we began to revise the design, we used MetaCricket techniques. Our second tag includes features similar to the first (a two-digit numeric display, the bicolor LEDs, a knob, and two push buttons), but it is based on the standard "blue dot" Cricket firmware, unmodified from its form in a conventional Cricket device. All of the specialized hardware is handled by a separate PIC processor, which communicates with the Cricket firmware as a bus device (Figure 15).

When Colella and fellow researchers design various simulation games, the tags are preloaded with special libraries for handling the communication between the tags, the events from any of the bus sensors, and the commands for generating visual or audible feedback.

Using MetaCricket, the design of this device was conceptually easier; technically it was much easier. The Cricket firmware did not change at all, and the firmware for controlling the specialized hardware was developed and debugged independently.

Figure 16 Stack of three tangible program bricks

Tangible program bricks. Tangible program bricks are 6×2 LEGO brick stacks with electronics embedded so that the bricks can communicate with one another and reveal their order and identity. The bricks were developed by Tim McNerney to explore tangible programming approaches and to make ideas of programming accessible to everyone.[16]

In developing tangible program bricks, McNerney took a PIC processor running the Cricket firmware and optimized it around communication. McNerney added a second bus communications pin, so that the bricks have one bus line running up the stack and one running down the stack. He added subroutines into the firmware for communicating over this second bus.

Figure 16 shows a stack of three program bricks. Inside each brick, a three-layer board stack holds electrical connectors facing up and down, the PIC microprocessor, and the serial memory. The top layer is a female "smart card" connector, consisting only of metal pads on a printed circuit board. The middle board has the PIC microcontroller and a male connector on the bottom. This connector has eight gold-plated springs that press against the printed circuit on the female connector in the expansion cards that can be inserted into the bricks. The bottom board contains the serial EEPROM chip and a connector that mates with the top of the brick below. The connector system passes power, bus communication (between neighbors), and a start/run signal to the entire stack of bricks.

Electrically, the program bricks are like Crickets with the standard peripherals (beeper, infrared communications, sensor inputs) removed. However, the program bricks include an expansion connector that allows various accessories to be plugged in. For instance, when an IR transceiver card is plugged in, the program brick can communicate as a Cricket and accept program download. If a beeper card is plugged in, a program brick can sound tones. Other accessory cards already developed include a capacitive touch sensor, a bus port for interfacing with standard bus devices, and an EEPROM card for storing numeric constants.

With this infrastructure in place, new programming paradigms can be explored and the system becomes more than a collection of Crickets. The program bricks are particularly useful for applications with a small set of primitives. For example, as a part of the "Counter Intelligence" project at the MIT Media Lab, the bricks were used to program microwave ovens. In another application, the bricks were used to construct dances for the dance craze buggies project.

Discussion

Applications like the ones we describe could be accomplished with other development tools; however, there are distinct advantages to MetaCricket. Generally, MetaCricket allows designers who are not engineers to incorporate features, possible only with digital technology, into their designs. To make this point very explicit, consider the simplest project presented in this paper, the electronic dog (Figure 2). "Dr. Martin" responds to varying light levels, turning its motors on and moving forward in response to a bright light. Consider alternative methods of implementing this idea:

1. *Analog circuit design.* An analog electronics engineer could design a transistor circuit. When the photocell level crossed a threshold voltage, a timer could be triggered to engage motor drivers for a hard-wired period of time.

2. *Microprocessor circuit design.* A computer engineer could design a microprocessor circuit that could run a program similar to the dog's Cricket Logo control program. Analog-to-digital converters would be used to provide the photocell signal. Motor driver circuits would be needed to drive the motors.
3. *Tethered interface.* Various desktop computer-to-sensor/actuator interfaces are available. Sensors and motors plug into the interface; a control program running on the desktop computer takes the sensor readings and issues motor commands. Some of these systems are as easy to use as the Cricket, but all require cable harnesses running between the interface and the project. For any sort of mobile or embedded project, these wires are both a practical and conceptual nuisance.

None of these alternatives is as appealing as the MetaCricket design. The first option requires sophisticated analog electronic skills. The second has recently improved with devices like the PIC microprocessor, which makes it substantially easier to design computational devices. But both options still require complex circuit design, electrical assembly, and programming skills. The third option (a tethered interface), especially in its educational versions (e.g., LEGO Dacta ControlLab), does allow prototyping by designers who are not experts, but the resultant product often has too many shortcomings, the most critical of which is the cable tether.

Of commercially available systems, the closest to the Cricket is the BASIC Stamp** programmable microcontroller.[17] The Stamp shares many similarities with the Cricket. It has a similar virtual machine and a development language designed to be easy to learn and use. So indeed, the Stamp and the Cricket are both designed with the idea of making prototyping easier for designers.

There are some important differences, however, between the Cricket and the BASIC Stamp. First, the BASIC Stamp's compiler is of traditional design: the user writes code, sends it through the compiler, downloads it to the project, and then runs the code. There is no possibility for interacting with the code after it is downloaded.

Next, the Cricket includes some very useful features as built-ins—namely, motor control and infrared communications—that require circuit design to be used with Stamp. Finally, the Cricket's bus system is simpler to use and is more flexible. Bus devices may be easily plugged in and removed from the system being developed, and the devices themselves are packaged in a more modular fashion. In short, one does not need to pick up a soldering iron to use the Cricket.

Future directions

The MetaCricket kit is designed for designers and helps them to implement a broad range of ideas. To this end, we are improving the kit in many ways. Our current research on MetaCricket focuses on expanding the kit to include a variety of new display devices, actuators, and sensors to allow designers to design and implement interfaces with new interaction modalities. We are expanding the communication capabilities of the Cricket. We are introducing new programming environments, particularly some on handheld-devices. This section briefly describes work in each of these areas.

Sensing and actuation. In collaboration with other research groups, we are working on new classes of sensors, including inductive and capacitive sensors, electric field sensors,[18] tag readers,[19,20] vision systems, and inertial measurement units (shock, orientation, spin, and position sensors).[21] On the output side, we are interfacing to new actuators and displays, including electromagnetic linear drive motors, solenoid actuators, E Ink displays,[22] and general-purpose LED, LCD (liquid crystal display), and fluorescent displays.

The Cricket architecture and design ideology has provided the foundation of other research projects. Using Crickets, Michael Eisenberg's Craft Technology Group has built special-purpose devices that embed computation into familiar form factors like the thumbtack and the hinge. In essence, these projects collapse several components of the MetaCricket system (motor, mechanism, and programmability) into a single object.[23] These craft technologies further lower the entry barrier for designers who wish to build and program computationally augmented objects.

Communication. We are expanding the communication capabilities of the Cricket to address differing design criteria. For example, infrared bus devices are ideal for applications requiring line-of-sight communication. Ultrasonic communication may be employed for applications without line-of-site requirements with the advantage that it will remain confined to a room, and RF and spread-spectrum communi-

cation are suited for long-range and multidevice applications.

We are also developing communication libraries that would simplify the design of interactive toys such as the musical fireflies. In designing many of the communicating toys described in this paper, we worked closely with the designers to implement the necessary low-level communication, handshaking, and error-correction procedures. In future versions of our software, we plan to provide a set of high-level primitive functions for this purpose.

New programming environments. In order to allow debugging and reprogramming devices built with the MetaCricket kit, we are developing new graphical and text-based (pen-based) programming environments for hand-held devices such as Microsoft Windows** Pocket PC machines, Palm** computers, and GameBoy** systems. We see these environments as especially suited for use by children and in space-limited classrooms, as in the typical elementary school.

Along a complementary direction, in collaboration with the Aesthetic and Computation Group at the Lab, we are integrating the MetaCricket kit with John Maeda's Design by Numbers (DBN) environment, which gives visual designers expressive control in a computational medium.[24] By joining DBN to MetaCricket, designers can build novel interfaces and allow real-world sensing to influence their visual displays. As Design by Numbers aims to provide visual designers with computational expression, MetaCricket will give industrial designers the capability to prototype fully functioning objects. We expect these kinds of tools to become commonplace in schools of design worldwide.

Conclusion

Perhaps the most unexpected outcome of MetaCricket is how it allowed us to leverage others' work. This was first demonstrated with the MidiBoat music synthesizer. After we integrated this device into MetaCricket, music became a central feature of Cricket projects, and researchers around our laboratory chose to use Crickets simply because it let them use the MidiBoat device. Not only did the use of the MidiBoat extend far beyond the intentions of its designers, but it also fostered the growth of the MetaCricket approach throughout our lab.

This particular story points to the ultimate power of MetaCricket. As the collection of bus devices grows, an exponentially increasing set of combinations becomes possible. This will in turn encourage more designers to use MetaCricket, and stimulate further growth of both application examples and development of new hardware "primitives." MetaCricket suggests a world where all devices share a common protocol, and can be reconfigured, repurposed, and reprogrammed by end users of all backgrounds and design styles.

As digital tools enter every aspect of our lives and the many artifacts of our world, it becomes more and more important that these objects are created from the sensibilities of designers. For instance, it seems inevitable that appliances in the kitchen of the future will talk to each other. MetaCricket will ameliorate this in two crucial ways. First, it will allow creative designers to prototype fully functioning digital behaviors, and thereby deeply influence the human qualities of final products. Second, it points to a future where end users will be able to redesign the functionality of the devices in their lives. We are looking forward to a day when the functionality of appliances and other manufactured articles is transparent, and consumers can reconfigure the technology around them in novel ways.

Acknowledgments

Robbie Berg has been an integral member of the Cricket team and has made many contributions to the project. We have greatly benefited from a team of undergraduate and graduate students working with us, including most especially Jan Malášek, Genee Lyn Colobong, Michael Rosenblatt, Ava Erickson, and Matthew Debski. We would like to thank the designers who have used our system. "Dr. Martin," the electronic dog, was created by Lene Camilla Holten. The beach ball remote car was created by Anthony Scelfo. Dan Overholt first showed us the Play-Doh synthesizer. Finally, we would like to acknowledge the ongoing encouragement, support, and insights of Mitchel Resnick throughout the Cricket project.

Cited references and notes

1. M. Schrage and T. Peters, *Serious Play: How the World's Best Companies Simulate to Innovate*, Harvard Business School Press, Cambridge, MA (1999).

2. F. G. Martin, *Children, Cybernetics, and Programmable Turtles*, master's thesis, MIT, Cambridge, MA (1988).
3. F. Martin, B. Mikhak, M. Resnick, B. Silverman, and R. Berg, "To Mindstorms and Beyond: Evolution of a Construction Kit for Magical Machines," *Robots for Kids: Exploring New Technologies for Learning*, A. Druin and J. Hendler, Editors, Morgan Kaufmann Publishers, San Francisco, CA (2000).
4. PIC microprocessor, see http://www.microchip.com/.
5. The primary difference in later versions of the PIC microprocessor was more internal RAM, which allowed us to move from an 8-bit to a 16-bit representation for numbers and program addressing.
6. F. G. Martin, *Circuits to Control: Learning Engineering by Designing LEGO Robots*, Ph.D. thesis, MIT, Cambridge, MA (1994).
7. J. Smith and J. Strickon, The MiniMidi Embedded Music Platform (aka The MidiBoat), see http://www.media.mit.edu/~jrs/minimidi.
8. Musical shapers and toys, see http://www.media.mit.edu/hyperins/projects.html.
9. T. M. Lackner, K. Dobson, R. Rodenstein, and L. Weisman, "Sensory Puzzles," *CHI 99 Extended Abstracts*, ACM Press, New York (1999), pp. 270–271.
10. G. Weinberg, B. Mikhak, and F. Martin, "The SqueezeMans: Interactive Handheld Musical Instruments," available from http://www.media.mit.edu/~gili/research/research.html.
11. G. Weinberg and T. Lackner, "The Musical Fireflies: Learning about Mathematical Patterns in Music Through Expression and Play," to be published in *Proceedings XIII Colloquium on Musical Informatics*, L'Aquila, Italy (September 3–5, 2000).
12. R. Borovoy and F. Martin, "The Dance Craze Buggies: A Tradeable Bits Technology," see http://el.www.media.mit.edu/people/borovoy/cars/.
13. Pogs are collectible disks that can be used to play a game similar to marbles. The name comes from a blend of passion fruit, orange, and guava juice sold by the Haleakala Dairy, whose colorful bottle caps were popular with children in Hawaii.
14. R. Borovoy, M. McDonald, F. Martin, and M. Resnick, "Things That Blink: Computationally Augmented Name Tags," *IBM Systems Journal* **35**, Nos. 3&4, 488–495 (1996).
15. V. S. Colella, *Participatory Simulations: Building Collaborative Understanding Through Immersive Dynamic Modeling*, master's thesis, MIT, Cambridge, MA (1998).
16. T. S. McNerney, *Tangible Programming Bricks: An Approach to Making Programming Accessible to Everyone*, master's thesis, MIT, Cambridge, MA (1999).
17. The BASIC Stamp, see http://www.parallaxinc.com/.
18. J. R. Smith, *Electric Field Imaging*, Ph.D. thesis, MIT, Cambridge, MA (1999).
19. R. Fletcher, *A Low-Cost Electromagnetic Tagging Technology for Wireless Identification, Sensing, and Tracking of Objects*, master's thesis, MIT, Cambridge, MA (1997).
20. J. Paradiso and K.-Y. Hsiao, "Swept-Frequency, Magnetically-Coupled Resonant Tags for Realtime, Continuous, Multiparameter Control," *CHI 99 Extended Abstracts*, ACM Press, New York (1999), pp. 212–213.
21. E. Hu, *Applications of Expressive Footwear*, master's thesis, MIT, Cambridge, MA (1999).
22. Electronic Ink, see http://www.eink.com/.
23. G. Blauvelt, T. Wrensch, and M. Eisenberg, "Integrating Craft Materials and Computation," *Proceedings of the Third Conference on Creativity and Cognition*, Loughborough, England (October 11–13, 1999), pp. 50–56.
24. J. Maeda, *Design by Numbers*, MIT Press, Cambridge, MA (1999).

Accepted for publication May 19, 2000.

Fred Martin *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: fredm@media.mit.edu).* Dr. Martin has been fascinated with computation since he was a small child. As a teen he longed to connect computers to the physical world—a passion he was able to fulfill as a computer science undergraduate at the Massachusetts Institute of Technology. He joined MIT's Media Laboratory in 1986, where he had the privilege of pursuing his doctoral work, supervised by Seymour Papert and Edith Ackermann. Dr. Martin co-created the MIT LEGO Robot Design Competition, a project-oriented course used worldwide as a model in undergraduate engineering education. He worked on a series of "programmable bricks" for young learners, which led directly to the LEGO Mindstorms Robotics Invention System product. Mr. Martin's textbook, *Robotic Explorations: An Introduction to Engineering*, will be published by Prentice Hall in 2000.

Bakhtiar Mikhak *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: mikhak@media.mit.edu).* Dr. Mikhak is a research scientist at the MIT Media Lab. His research interests include developing new programming environments and computational construction kits for science and engineering education, and for rapid prototyping by professional designers. For the past three years, he has worked closely with educators and children in classroom and after-school settings, and with designers from a number of Media Lab sponsor companies. Dr. Mikhak received the B.A. degree in physics from the University of California, Berkeley (1988) and M.S. and Ph.D. degrees in theoretical physics from the University of California, Los Angeles (1989, 1995).

Brian Silverman *MIT Media Laboratory, 20 Ames Street, Cambridge, Massachusetts 02139-4307 (electronic mail: bss@media.mit.edu).* Mr. Silverman divides his time among the MIT Media Lab, Metrowerks, Inc., and Logo Computer Systems Inc. (LCSI). Mr. Silverman was one of the founders of LCSI, the world's leading developer of Logo software. He directed the development of more than a dozen commercial educational software products (including LogoWriter, MicroWorlds, and the Phantom Fishtank), many of which have won major awards from industry groups and publications. As a visiting scientist at MIT, he has been centrally involved in the development of StarLogo, Programmable Bricks, and Crickets.