First Results with eBlocks: Embedded Systems Building Blocks

Susan Cotterell, Frank Vahid, Walid Najjar, Harry Hsieh
Department of Computer Science and Engineering
University of California, Riverside
{susanc, vahid, najjar, harry}@cs.ucr.edu
*Also with the Center for Embedded Computer Systems at UC Irvine

Abstract

We describe our first efforts to develop a set of off-the-shelf hardware components that ordinary people could connect to build a simple but useful class of embedded systems. The class of systems, which we call monitor/control systems, is composed primarily of sensors – light, motion, sound, contact, and other types – and output devices – light-emitting diodes, beeping speakers, or even electric relays that control electric appliances like lamps. For example, one monitor/control system would detect if a house's garage door was open at night, and would blink an LED inside the house to alert the homeowner of this normally undesirable situation. Today, configuring even the most basic monitor/control system requires knowledge of electronics and programming. We seek to create a set of building blocks, which we call eBlocks, that would enable someone with no knowledge of electronics or programming to be able to build simple but useful monitor/control systems. We are creating eBlocks largely by incorporating intelligence into previously dumb sensors and output devices, and by developing a set of standards and methods that enable eBlocks to work together seamlessly when connected. eBlocks have only recently become possible due to the extremely low cost, low power, and small size of embedded microprocessors. We describe our first results of creating a basic class of eBlocks, Boolean eBlocks, that from a user's perspective transmit or receive only "yes" or "no" signals. We discuss the internal eBlock design, eBlock system design issues and decisions, and several eBlock-based systems designed by ourselves and by undergraduate students.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]

General Terms

Design, experimentation, human factors.

Keywords

Embedded systems, intelligent homes, networks.

1. Introduction

Home and office environments today typically include numerous embedded computing systems, such as burglar alarm, temperature control, and remote appliance control systems. Those systems are designed by engineers. However, there is a class of useful embedded systems that ordinary people with no engineering training could build themselves if only the right building blocks existed.

We refer to that class of systems as simple *monitor/control systems*. A monitor/control system senses events, such as a person walking across a room, a door being opened, or a button being pressed. Such a system then responds by generating outputs, such as blinking a light-emitting diode (LED), sounding a beep, or turning on an electric relay that controls an electric appliance like a lamp. We focus on human-scale monitor/control systems, referring to events and outputs that a person could observe or generate, as distinguished from other scales of events or outputs, like detecting or generating radio signals.

Human-scale monitor/control systems are grossly under-represented in existing commercial embedded system products. Some applications are available as products, like motion-sensing lights or timer-controlled relays, but these products are very limited in their function and cannot easily be extended or customized. The main reason for the under representation is cost: most applications are too specialized to be cost effective when one considers the many real costs (packaging, marketing, store placement, etc.) of introducing a new consumer product into the market.

Consider the following potential application. New homeowners often forget to close their garage door at night. A useful application would blink an LED inside the house if this condition occurs. Such an application could be built using a light sensor, a magnetic contact switch, a microcontroller that computes a logic function of those two inputs, a wireless transmitter, a wireless receiver, another microcontroller, and an LED. While seemingly simple, building a working system from these components is beyond the skills of ordinary people, and even a challenge for most engineers who haven't been specifically trained in embedded system design. Yet, companies generally could not profit from such a product, since most people are not willing to spend more than perhaps $20-$40 on such a product. People willing to spend much more than that would instead purchase a home alarm system.

Similar applications are plentiful. A storeowner may want to detect a customer at the front counter and sound a beep in the storeroom. A dog owner may want to display if the backyard gate is left open. A cafeteria manager may want a simple way for service line workers to indicate to kitchen staff which food items need replenishing. A hard-of-hearing person may want a small vibrating device that indicates noise (perhaps from a crying baby). A classroom teacher may want students to be able to anonymously vote on various subjects by pressing buttons at their desks. A farm owner may want to detect if there is mail in his remotely located mailbox. A carpool driver may want a way other than honking the horn to inform a passenger that the car is outside the passenger's

apartment. All of these applications are useful, but none by itself is in high enough demand to warrant a stand-alone product, and the aforementioned applications have enough differences that prevent a single product from covering all the applications.

We are developing a set of building blocks from which ordinary people could easily implement these and many other applications, without any training in electronics or programming. We refer to these blocks as *eBlocks*, or embedded system building blocks. In this paper, we first discuss related work in Section 2. In Section 3, we describe the subset of eBlocks that we have addressed so far, known as Boolean eBlocks. Section 4 highlights numerous challenges and solutions in designing Boolean eBlocks. Section 5 describes several useful monitor/control systems that we and undergraduate students have designed using Boolean eBlocks. Section 6 discusses eBlock prototypes and the feasibility of battery based eBlocks. Section 7 describes experiences testing eBlocks on various users. Section 8 provides conclusions and points to numerous future directions.

2. Related Work

Much work has been done to simplify and make computing devices more accessible. MIT's Beyond Black Boxes project [6] focuses on providing tools and materials for science education. MIT Crickets, evolved from the Programmable Bricks project [14][15] and Thinking Tag project [2]. A cricket is a tiny computer, powered by a 9-volt battery, that can receive information from two sensors and can control two motors. Crickets are programmed utilizing the Logo language [17][20] – a simple, graphical, highly intuitive language. In fact, Crickets provided the foundation for the Lego Mindstorms [25], which include numerous sensors and actuator Lego blocks that are connected to a central microprocessor block to build a variety of small robots. The central microprocessor block is programmed using a simple graphical language included with the kit. While the Beyond Black Boxes projects seek to motivate people to program crickets, eBlocks seek to enable people to build systems. However, we can of course introduce eBlocks that could be programmed by "advanced" users, and we suspect such programming would build upon the Logo language.

Home automation is another area in which much work on ubiquitous computing has been done. In home automation, an emphasis is placed on interoperability between higher-end consumer electronics, such as audio, video, and other media content, and the PC. However, many problems still exist. A consumer will encounter systems that are complex to set up and maintain. There are a myriad of issues facing consumers discussed in [9] including knowledge of networking, media management, security, and content protection. Such issues quickly overwhelm the majority of the population.

To address some of these problems, a framework to support home interoperability is presented in [19]. A network exists in which devices have the ability to discover, configure, and control other devices on that network. Network devices range from a PC, television, stereo, or a variety of consumer electronics. This type of system eliminates the need for a user to configure and control each device independently on the network. The setup, configuration, and control aspects of the system are transparent to the user and make the system more feasible. There is no need for a professional to setup a device every time a new device is added to the system.

To guarantee interoperability between devices on the network a common protocol is required. The Universal Plug and Play

Figure 1: Smart Dust

Courtesy of Joe Kahn

Forum (UPnP) [22][24] is an emerging industry initiative that is trying to make connectivity between stand-alone devices and PCs simple and consistent. They are currently working on defining standards and templates for classes of devices so that when an individual device is added to the network it can easily be integrated, regardless of how the individual devices are implemented because a common interface exists. Home interoperability and UPnP devices are typically complete applications, not building blocks, yet we expect to eventually develop an eBlock that interfaces to a UPnP device.

X10 [26] is another emerging protocol for compatible devices throughout the home to communicate via existing 110V wiring in the house. X10 superimposes a (digital) signal onto the power signal, without interfering with the power signal. X10 devices detect the superimposed signal and look for their ID number, and then react to commands to turn on or off. A common application of X10 is to control all the lights or power appliances of a home from a single master device.

Smart sensors [13][18][23] embed a sensor's data sheet information directly in the sensor device, typically in a non-volatile memory like EEPROM. This embedded information is known as a Transducer Electronic Data Sheet, or TED and includes information such as the manufacturer ID, model number, sensor's use, calibration information, voltage levels, and temperature ranges. The embedded information relieves a system designer from having to manually enter the information into their software and enables some automatic configuration. IEEE has developed several standards related to such basic smart sensors, known as IEEE P1451 [12]. In addition, advanced smart sensors seek to perform processing in sensors themselves, primarily to reduce the amount of network data traffic [3]. Sensors which have added intelligence, are able to perform tasks such as conversions, monitor machinery for wear, perform image compression, monitor neighborhoods for real-time pollen count, or even spy plane applications, as discussed in [1][11][21].

There has also been work done in miniature wireless sensing devices [4][7][8], referred to as the Mica wireless platform. These devices have sensing, communication, and I/O capabilities and are intended to last years in the field utilizing only a pair of AA batteries. Each Mica node consists of processor/radio circuits that are sandwiched together with sensor circuits. A variety of sensor circuits such as temperature, magnetic field, light, acceleration, vibration, and acoustics are available. A system designer would customize the Mica node to their particular application by selecting which sensors are incorporated. A collection of Mica nodes are capable of self-configuring a multi-hop network, utilizing RF communication, and support dynamic reprogramming within the network. The nodes also contain the TinyOS operating system and allow designers to customize communication

protocols to suite their networks. The newest generation of these wireless platforms is Smart Dust [5][26], which are on the millimeter scale in size as shown in Figure 1 – illustrating just how small devices like eBlocks could eventually become. Smart Dust utilizes thick film batteries, a solar cell, or both for power, providing roughly 1-2 Joules per day. These devices share many of the characteristics of the Mica nodes but utilize optical communication and have more restrictive power utilization limits.

The field of sensor networks in general focuses on coarse-grained network-level applications. In addition, the Mica and Smart Dust platforms require knowledge of the underlying architecture and communication, programming, operating systems, networking, etc. Unfortunately, ordinary people do not have such expertise. While some of the sensing capabilities of Mica and Smart Dust are similar to eBlocks, the internal design and communication interface requirements for eBlocks are vastly different, as eBlocks are intended for a different audience, namely people with no engineering experience. While designing a low-power, self-configuring device is useful in developing eBlocks, we must incorporate such features without requiring any additional effort by the user.

3. Defining Basic eBlocks

We initially defined a basic set of eBlocks from which a variety of monitor/control systems could be built. A principle guiding the definition task was that eBlocks should be intuitive and easy to understand by nearly anybody old enough to want to build such a system. In particular, we assumed no engineering or technical training from the users, and no training in logic. We sought for eBlocks to be usable almost immediately with little training, and to be usable in remote locations without access to a centralized computer. Based on these requirements, we early on determined that eBlocks should be decentralized and should work just by connecting them together.

3.1 eBlocks that Operate on Boolean Values

We sub-divided the basic set of eBlocks into three categories:
1.   Sensor blocks – These blocks detect environmental events of interest, such as motion, light, sound, or contact.
2.   Output blocks – These blocks generate events for observation or control, such as lighting an LED, sounding a tone, or controlling an electric relay.
3.   Communication/logic blocks – These blocks assist with communication among sensor blocks, supporting wireless transmission, wireless reception, boosting of signals over long wires, storing of signals for certain durations, and logically combining multiple signals into a new signal.

After initially exploring potential definitions of eBlocks and creating hypothetical eBlock-based systems, we realized that the challenge was enormous. We thus decided to first focus on a restricted but still very useful subset of eBlocks that transmit and receive Boolean values only, as opposed to also including integer values or other types of values. We refer to such blocks as *Boolean eBlocks*.

Boolean eBlocks presented an initial challenge of choosing the most intuitive representation of "true" and "false." We found that ordinary people are not particularly comfortable with the notions of "true" and "false" in the context of monitor/control systems. (In contrast, to a reader of this paper, true and false are likely obvious concepts, as the reader likely has some experience

using Boolean logic).   After considering numerous possible representations, such as true and false, 1 and 0, high and low, on and off, etc., and presenting different possibilities to non-engineers of various ages, we settled upon *yes* and *no* as the most intuitive for eBlocks. For example, we describe a motion sensor by asking the question: "Is motion detected? *yes* means motion is detected, *no* means motion is not detected."

3.2 A Basic Boolean eBlock Catalog

We sought to develop a catalog of Boolean eBlocks that balanced the conflicting goals of having the fewest possible eBlocks, of having eBlocks that were intuitive, and of having eBlocks that required minimal configuration expertise. At one extreme, we could have a single eBlock that could do anything, but that eBlock would likely be big, power-hungry, and require extensive programming. At the other extreme, we could build a unique eBlock for every possible component, but that would result in an intimidating large catalog too big for a user to readily comprehend, and systems that required too many basic components for users to reasonably build.

We developed our eBlock catalog by considering a dozen applications, many of which we listed in the introduction. We started with a minimal set of eBlocks, and introduced new eBlocks, or added features to an existing eBlock, only when building an application with the minimal set seemed to be too difficult for an ordinary user.

Below are the eBlocks we defined.

- Sensor eBlocks:
  - Magnetic Contact Switch - detects when contact between two sensors is made
  - Light-Beam Switch - composed of a light source and light sink, this device detects when the light beam is broken
  - Motion Sensor - detects the presence of motion
  - Light Sensor - detects the presence of light
  - Sound Sensor - detects the presence of sound
  - 3-Key Entry - detects if a predetermined sequence of keys is pressed correctly
  - 10-Key Entry - detects if a predetermined sequence of keys is pressed correctly
  - Button - detects when a button is pressed
- Output eBlocks:
  - Green/Red LED - blinks a green light when input is *yes*, blinks a red light when input is *no*
  - Blinking LED - blinks a light when input is *yes*
  - Beeper - emits a beeping sound when input is *yes*
  - Electric Relay - transmits electricity when input is *yes*
- Communication/Logic eBlocks:
  - Pulse Generator - outputs *yes* and *no* pulse where the *yes* time and *no* time is user defined
  - Clock Timer - user sets pins to indicate at which times to toggle the output over a 24-hour period
  - Splitter - receives a signal and replicates that signal on each output
  - Toggle - input of *yes* toggles the current value outputted by the device
  - Prolonger - input of *yes* causes output to become *yes*, output resets to *no* when the device times out

- Wireless Transmitter - wirelessly transmits a signal to another eBlock
- Wireless Receiver - wirelessly receives a signal from another eBlock
- 2-Input Logic Block - configurable logic block programmed by the user via DIP switch
- 3-Input Logic Block - configurable logic block programmed by the user via DIP switch
- Yes/No Block - outputs a constant *yes* or *no* depending on user defined setting

Table 1 includes a more detailed description for a subset of eBlocks used in later examples.

4. Design Issues

We now describe several challenges and solutions during the design of eBlocks.

4.1 Logic Blocks

A key challenge was finding a way to create an eBlock that would generate an output that is a logical expression of the block's input values. The challenge lied not in technical issues, but rather user interface issues. In the garage door example in the introduction, we want to blink an LED when the garage door is open and outside is dark. Our sensors would be a magnetic contact switch that outputs *yes* when the door is closed (let's call this signal A) and an outdoor light sensor that outputs *yes* when light is detected (signal B). The correct logical expression would be $A'B'$, or the door is not closed and light is not detected. However, this simple exercise of creating the correct Boolean logic expression is well beyond the skills of ordinary people. For example, when we began to explain the logical expression approach to a group of potential users, one of the users interrupted to say "I have *no clue* what you are talking about." Again, the reader of this paper is likely *not* an ordinary person in the context of comfort with Boolean logic. The task becomes even more difficult if the user wants to detect multiple conditions, e.g. $AB + A'B'$. Thus, creating a logic eBlock

that requires the user to enter a Boolean expression is undesirable. Some studies show that most people can't form basic logical expressions, and this largely explains why such expressions are almost completely unused in common applications like Internet search queries [10].

We observed potential users naturally trying to figure out the function of a logic eBlock by enumerating the input possibilities and deciding the appropriate output for each possibility – in other words, creating a truth table. After considering numerous options and observing potential users, we developed the following initial solution to the challenge. We provide two and three input logic eBlocks. For a two-input logic eBlock, we provide a four-switch DIP (dual-inline package) switch. Each switch can be moved to a *yes* or *no* position. The eBlock has two inputs A and B. Each of the four switches correspond to a particular combination of input values, as follows:

A	B	Output
no	no	n ▢ y
no	yes	n ▢ y
yes	no	n ▢ y
yes	yes	n ▢ y

This simple table can be printed directly on the eBlock package next to the DIP switch. Likewise, an eight-pin DIP switch would be used for a 3-input logic eBlock. This solution has the benefit of minimizing the translation of the input values to output values and avoids "encoding" the logic expression in a Boolean function. The solution involves having the user simply provide the appropriate output for each possible input, something users tended to try to do anyways.

A limitation of the above solution is that the user must still translate sensor outputs into variables A and B. Furthermore, the solution does not extend well to more than 3 inputs – we currently require the user to manually use multiple logic blocks for more inputs.

Table 1: Partial Boolean eBlock Catalog

	eBlock	Diagram	Description	Interface
Sensor	Magnetic Contact Switch		Determines when contact between two sensors is made.	yes = contact between sensors no = no contact between sensors
Sensor	Light Sensor		Sensor detects presence of light.	yes = light detected no = no light detected
Sensor	Button		Indicates whether button is pressed or not.	yes = button pressed no = button not pressed
Output	LED		Device blinks a light when input is a yes. Device emits no light when input is no.	yes = blink LED no = turn LED off
Communication/logic	Splitter		Device receives a signal and replicates that signal on each output.	yes = output yes signal no = output no signal
Communication/logic	Toggle		An input of yes toggles (inverts) the current value outputted by the device.	yes = toggle previous output value no = do nothing
Communication/logic	2-Input Logic Block		Configurable logic block programmed by the user via DIP switch.	For each of the possible outcomes of a and b, there is a corresponding switch which can be set so the resulting output is a yes or no for that particular combination.
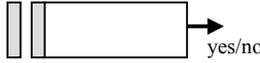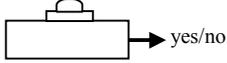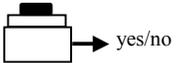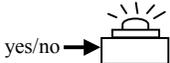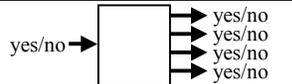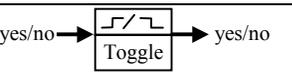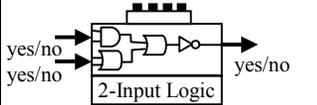
Figure 2: eBlock Internals (a) Sensor Block – Light Sensor eBlock (b) Communication/Logic Block – 2-Input Logic eBlock (c) Output Block – Green/Red LED eBlock

(a)   (b)   (c)
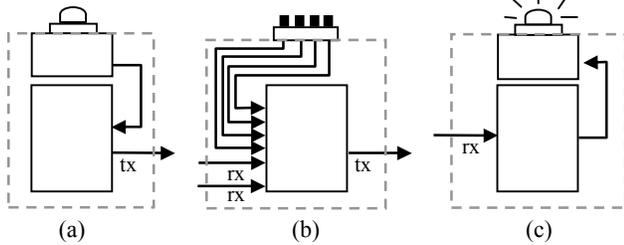
4.2 Computers in Every Block

We quickly determined that communication of logic values among eBlocks could not be implemented directly as a physical wire carrying 1 for *yes* and 0 for *no*, for several reasons. First, for low power, we did not want to transmit a signal at all times. Transmission of the output signal is one of the most power-costly operations of a block. Yet, we want eBlocks to potentially be powered by small batteries like a watch battery. Thus, we want to transmit as infrequently as possible. Second, glitches are common among sensors, yet we do not want to trigger outputs unless real events are detected. Third, we may want to transmit a message indicating an *error* rather than *yes* or *no*. Fourth, we may in the future want to transmit information about the sensors to assist the user in configuring logic blocks. Fifth, we want to support the option of wireless transmission.

Thus, we determined that blocks should communicate packets, not just 1 and 0 for *yes* and *no*. Physically, the blocks are connected via a single serial line. That single line carries serially transmitted packets. A packet consists of a start bit, followed by two bits indicating *yes/no/error*, followed by several stop bits. In the future, we will incorporate identification information into the packet. A block can also transmit no signal at all. Blocks with inputs from other blocks can use a pull-up resistor circuit to convert a situation in which these is no input signal into a stop bit.

To support packets, every block must contain a simple computer, as shown in Figure 2. A sensor block converts its physical input signal into a packet that the block sends over the serial output. An output block receives packets and decodes them into a physical output. A communication block receives an input packet and either changes the block's internal state, or sends out an identical output packet. A logic block receives input packets, computes the logic output, and transmits a new output packet.

Clearly, the compute requirements are modest. We therefore

have chosen to use a PIC microcontroller, which is a popular 8-bit microcontroller distinguished for very low costs of less than a dollar (in quantity), very low power measured in microwatts (whereas milliwatts is more typical of other microcontrollers), and small size. The microcontroller typically includes on-chip non-volatile program memory and several peripherals, like analog-digital converters and serial transmitters/receivers (UARTs). Such microcontrollers are found in a myriad of common low-cost low-power items, such as sneakers with blinking LEDs.

The computer in every block and standard communication features are what distinguishes eBlocks from existing off-the-shelf sensor, output, and communication blocks.

4.3 Event Granularity

The rate at which eBlocks send packets must be selected carefully. The rate should be variable so eBlocks can reduce the rate to reduce power. Yet the fastest rate must be known to all eBlocks so that one eBlock does not send packets faster than another can process those packets. The slowest rate should not be so slow as to create annoying time lag in the system, and should be known to all eBlocks so a failed eBlock can be detected. We examined rates in light of their impact on power and found that rates faster than around 50 milliseconds yielded too short of battery lifetimes while rates slower than a few seconds yielded unacceptable delays in some systems. Thus, we have presently set the fastest rate at 50 milliseconds and the slowest at 3 seconds. We found 50 milliseconds to be plenty fast for nearly all sensing activities -- in fact, such speed is hardly ever really necessary. We found 3 seconds to introduce a slightly annoying lag in some systems having chains of about 6 eBlocks (chains typically don't get much longer), resulting in a 15-20 second delay from input to output. That could be annoying for systems where a user pushes a button that should generate an output, but is no problem for systems that monitor slower events (like the Garage Door Open at Night Detector).

We anticipate adding a simple adjustment screw to eBlocks to allow a user to vary the rate from a default rate of about 1.5 seconds, to accommodate the need for longer battery or faster system response (note: precision is not important in selecting the rate). Such screw-based adjustment is already commonplace in certain sensors, like motion and light sensors.

Note that the packet send rate is independent of the baud rate. We require all eBlocks to communicate their packets at 1200 baud.

5. eBlock-Based Systems

Utilizing the eBlocks shown in Table 1, we can build a variety of systems. For example, Figure 3(a) illustrates how the Garage

Figure 3: eBlock Systems: (a) Garage Door Open at Night Detector (b) Cafeteria Food Alert.

(a)   (b)
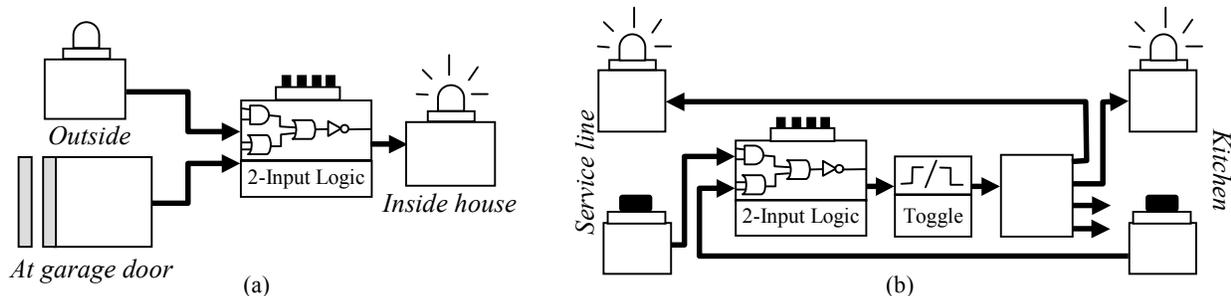
Figure 4: eBlock prototypes

Table 2: eBlock Prototype Estimated Battery Lifetimes with PIC Constantly Running (battery capacity = 19278 Joules).

eBlocks	Energy/day (J/day)		Lifetime
	PIC	HW	
Button	8.684	0	6 years
Light Sensor	8.684	1339.2	0.5 months
LED	8.640	64.8	9 months
Green/Red LED	8.640	129.6	5 months
Beeper	8.640	180.0	3 months
2-Input Logic Block	8.728	0	6 years
Toggle	8.684	0	6 years
Prolonger	8.684	0	6 years
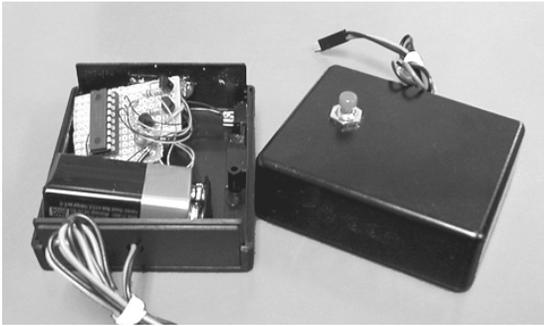
Door Open at Night Detector application, described earlier, could be built. A light sensor is utilized to determine when it is dark outside and a contact sensor is used to determine if the garage door is open. The outputs of these sensors are fed into a 2-input logic block. Because we are interested only when it is dark and the door is open, we want to output a *yes* from the 2-input logic block when the light sensor emits a *no* and the magnetic contact switch emits a *no*. We accomplish this functionality by setting dip switch 0 to the *yes* position.

Furthermore, using the same eBlocks shown in Table 1, a cafeteria manager can design a simple system for service line workers to indicate to kitchen staff which food items need replenishing. As shown in Figure 3(b), we start by placing an LED and button pair by the service line workers. When a food item runs low, the service line worker can simply press the button. The LED will start to blink indicating to the service line worker a request has successfully been made. A second LED and button pair is also located in the kitchen. When the service line worker presses the button, the LED in the kitchen will also blink, indicating that a food item is running low. When the food item has been replenished, either button can be pressed to turn both LEDs off. However, if there are numerous food items in the service line then workers in the kitchen do not know which item needs to be replenished. Thus, we can extend the design so that each food item has a LED/button pair associated with it. The kitchen could then have labels with LED/button pairs corresponding to each food item.

6. Simulation and Prototypes

6.1 VHDL

We have simulated sixteen eBlocks including the eBlocks used in the aforementioned examples in VHDL. We then built the systems shown in Figure 3 as well as ten others by combining the corresponding eBlock components in VHDL. We simulated those systems using Synopsys and verified them for correctness.

6.2 Prototypes

We next built physical prototypes, shown in Figure 4, using a PIC microcontroller, the corresponding circuitry for each of the various eBlocks, and a 9V battery. Currently we have the following eBlock prototypes implemented: Button, Light Sensor, Motion Sensor, 2-Input Logic, Toggle, Prolonger, LED, Green/Red LED, Buzzer, Electric Relay, and Magnetic Sensor eBlocks. On average, we implemented each of the various eBlock prototypes using 275 lines of C code. Communication between eBlocks was implemented using the internal UART of the PIC microcontroller at a transfer rate of 1200 baud. Furthermore, we

successfully tested how far eBlock communication was effective by connecting a button eBlock to an LED eBlock and testing various lengths of wire. Currently, we have exceeded a distance of 1 mile (approximately 6000 feet) using standard twisted pair wire. We plan to keep increasing the length of wire to determine at which point communication fails, however at such lengths we would expect a user to use wireless receiver and transmitter eBlocks. Using the aforementioned eBlocks prototypes, we implemented the Garage Door Open At Night and Cafeteria Food Alert system, discussed in Figure 3, as well as others.

6.3 Battery Lifetime

We want most eBlocks to be battery powered, to avoid the complexity of connecting every component to a wall power source. In addition, we must ensure that battery life is sufficient such that a user is not constantly changing batteries. We used an off-the-shelf 9V battery with a capacity of 19,278 Joules, which correlates with the goal of trying to keep eBlocks to be as small as possible. Table 2 shows the expected battery life for several of the various eBlock prototypes implemented. For each of the eBlocks we list the Joules consumed per day required by the PIC microcontroller in the *PIC* column, the Joules consumed per day for the corresponding circuitry in the *HW* column, and finally the expected lifetime of the prototype denoted in months or years in the *Lifetime* column.

We determined the energy consumption of the PIC microcontroller using the corresponding datasheet [16]. The PIC microcontroller consumes 20 μA when active and 0.20 μA when in power down mode. Furthermore, driving a PIC port high consumes 3 mA, and driving a PIC port low consumes 8.5 mA. We also consider the power of driving a wire up to 100 feet long, based on physical measurements of a standard low-cost wire. The PIC energy consumption, listed in the *PIC* column, assumes that the PIC sends a packet every 2.5 seconds and considers driving the output port and 100 feet of wire. Normally the PIC will send a packet every 3 seconds; however, we also wanted to consider packets sent upon a change of the eBlock's input.

Furthermore, we considered the energy consumed by the sensors and corresponding circuitry of each of the various eBlocks. To obtain the energy consumed by the circuitry of each of the various eBlocks we physically measured the current when the device was active. For example, we connected a multimeter to the LED eBlock, and measured the current when the LED was on. We found that the LED consumes 8.9 mA at 5 volts when on. Recall however that our LED eBlocks blink rather than being constantly illuminated, since a constantly on LED would drain a battery very quickly. We then took into consideration that the

Table 3: eBlock Prototype Estimated Battery Lifetimes with PIC Power Down Mode Activated and Low Power Components (battery capacity = 19278 Joules).

eBlocks	Energy/day (J/day)		Lifetime
	PIC	HW	
Button	2.722	0	20 years
Light Sensor	2.722	44.5	1 year
LED	2.678	14.4	3 years
Green/Red LED	2.678	28.8	2 years
Beeper	2.678	27	2 years
2-Input Logic Block	2.766	0	19 years
Toggle	2.722	0	20 years
Prolonger	2.722	0	20 years

LED eBlock blinks roughly every 3 seconds for about a tenth of a second and estimated that the status of the block would be *yes* half of the time. Thus, on average, the LED is illuminated a tenth of a second roughly every 6 seconds. We estimated the energy of each of the eBlocks that require additional hardware in a similar manner and listed the corresponding energy consumption in Table 2.

As shown in Table 2, for most eBlocks the 9V battery will provide enough energy to last several years. However, some eBlocks have a short battery life expectancy. For example, the light sensor eBlock is estimated to last less than a month and thus does not have a reasonable battery life. Currently, this eBlock will require a different power source.

In the future, we must either develop a different implementation using lower power components or consider a sampling approach in which we sample the light inputs at some specific interval. Table 3 show the estimated battery lifetimes for each of the corresponding eBlocks if we use power saving strategies such as powering down the PIC microcontroller when the PIC is idle, using lower power components, and sampling inputs. For example, lower power components were not available for the light sensor. Thus, we considered a sampling approach that monitors the light level at an interval of every 3 seconds. Instead of constantly powering the light sensing circuitry, we only need to power the circuit for a short duration while sampling. Otherwise, we can shutdown the circuitry, thereby significantly increasing battery life from less than one month to slightly longer than 1 year. Furthermore, we can add configurability to the light sensor, which allows the user to customize the sampling rate to their specific application. If a user is only interested in sampling the light level once every hour, the lifetime of the light sensor eBlock would be further increased to over 6 years.

7. Experiences with eBlocks

For several years, we have required a three-week project in an upper-division embedded systems university course. The project was similar in complexity to the garage door open at night project. The students already had several months of experience in programming microcontrollers, assembling basic electronic systems, implementing serial communication, and interfacing with some sensors and display devices. The project involved new sensors and display devices, and hence students had to find components in electronics catalogs and read datasheets to learn how to interface with those components. Of about 50 students who have attempted the project, only 20 were able to successfully complete the project in the three weeks. Most problems

encountered related to misunderstanding certain data sheets, errors during interfacing, and difficulty in debugging.

This year, we introduced a similar project but allowed the 22 students to use eBlocks, which were described using a simple 3-page catalog that included basic examples. The students needed less than 30 minutes to comprehend that material. All students successfully designed the garage door open at night system using eBlocks, in less than one hour (designs could be simulated in VHDL, but at the time we did not yet have a complete set of physical prototype blocks). Furthermore, students were given two hours more to create new designs, and they came up with numerous creative and useful applications.

8. Conclusions and Future Work

eBlocks greatly reduce the time needed to build basic monitor/control embedded systems and eliminate the need for programming or electronics expertise. eBlocks will not replace existing engineer-designed embedded systems, but will enable ordinary people, as well as a wider variety of engineers and programmers, to quickly realize a large range of new and useful applications.

We built physical eBlock prototypes and will use those in future courses as well as in local high schools (to observe truly non-experienced users). Immediate future work will include support to make logic configuration even simpler and extension to eBlocks that communicate integers rather than just Boolean values. As we discuss eBlocks with people in various fields, new applications that could be straightforwardly implemented with eBlocks continue to surface, such as detecting the speed of vehicles on a local street (requiring integer eBlocks), detecting if a child or hospital patient gets down out of a bed, detecting a water leak in a second home and calling the homeowner with a prerecorded message, and controlling a heat lamp and fan in a temperature-sensitive chemistry experiment. The potential list of applications is likely enormous.

9. Acknowledgments

10. References

[1] K. Aizawa. On Sensor Image Compression. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 7, No. 3, June 1997.

[2] R. Borovoy, M. McDonald, F. Martin, and M. Resnick. Things that blink: Computationally augmented name tags. IBM Systems Journal, Vol. 35, No. 3&4, 1996.

[3] M. Clarkson. Smart Sensors. Sensors Magazine, May 1997.

[4] Crossbow Technology Inc., http://www.xbow.com

[5] Dust, Inc. http://www.dust-inc.com

[6] Epistemology and Learning Group, MIT Media Laboratories. Beyond Black Boxes. http://llk.media.mit.edu/projects/bbb/

[7] J. Hill, D. Culler. MICA: A Wireless Platform For Deeply Embedded Networks. IEEE Micro, Vol. 22. No. 6, November/December 2002.

[8] M. Horton, D. Culler, K. Pister, J. Hill, R. Szewczyk, A. Woo. MICA The Commercialization of Microsensor Motes. Sensors, April 2002.

[9] Interoperable Home Infrastructure. Intel Technology Journal Vol. 6, Issue 4, 2002.

[10] B.J. Jansen, and U. Pooch. Web user studies: A review and framework for future work. Journal of the American Society of Information Science and Technology. 52(3), 2000, 235 – 246.

[11] C. Johnson. Smart Sensors Extend Web Scale. EE Times, April 2001.

[12] K. Lee. A Synopsis of the IEEE P1451 – Standards for Smart Transducer Communication. National Institute of Standards and Technology, 1999.

[13] T. R. Licht. The IEEE 1451.4 Proposed Standard And Emerging Compatible Smart Transducers and Systems. National Institute of Standards and Technology, 2000.

[14] F. Martin, et. al. The MIT Programmable Brick. http://lcs.www.media.mit.edu/groups/el/projects/programmable-brick/

[15] F. Martin, et. al. Crickets: Tiny Computers for Big Ideas. http://lcs.www.media.mit.edu/people/fredm/projects/cricket

[16] Micochip, http://www.microchip.com

[17] S. Papert. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, New York, 1980.

[18] D. Potter. IEEE P1451.4's Plug-and-Play Sensors. Sensors Magazine, December 2002.

[19] Y. Rasheed, J. Edwards, C. Tai. Home Interoperability Framework for the Digital Home. Intel Technology Journal, Vol. 6, Issue 4, 2002.

[20] M. Resnick, S. Ocko, and S. Papert, LEGO, Logo, and Design, Children's Environments Quarterly 5, No. 4, pg. 14-18, 1988.

[21] P. Saffo. Smart Sensors Focus on the Future. CIO Insight, April 2002.

[22] E. Steinfeld, Devices that play together, work together. EDN Magazine, September 13, 2001.

[23] B. Travis. Sensors Smarten Up. EDN, March 1999.

[24] Universal Plug and Play Forum. http://upnp.org/

[25] P. Wallich. Mindstorms Not Just a Kid's Toy. IEEE Spectrum, Vol. 38, No. 9, September 2001.

[26] B. Warneke, M. Last, B. Liebowitz, and K. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. Computer Magazine, pg. 44-51, January 2001.

[27] X10 protocol, http://www.x10.org/.