UNIVERSITY OF CALIFORNIA
RIVERSIDE

External Interfaces and Software Tools for Electronic Blocks

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Shawn Nematbakhsh

March 2005

Thesis Committee:
Dr. Frank Vahid, Chairperson
Dr. Harry Hsieh
Dr. Walid Najjar

ABSTRACT OF THE THESIS

External Interfaces and Software Tools for Electronic Blocks

by

Shawn Nematbakhsh

Master of Science, Graduate Program in Computer Science
University of California, Riverside, March, 2005
Dr. Frank Vahid, Chairperson

Electronic Blocks (eBlocks) are modular single-function blocks that can be assembled to build simple useful systems. In the past, eBlock systems were entirely self-contained, interfacing only with other eBlocks. In order to expand the possibilities for eBlocks, we introduce new blocks capable of interfacing with the telephone system, PDAs, and The Internet. In addition, we introduce an eBlock simulator that can be used to design and test systems before physical construction. We expand on the simulator to create an eBlock synthesis tool, allowing virtual systems in software to be downloaded onto a programmable eBlock. Our results show that external eBlock interfaces combined with our synthesis tool produce better, simpler, and more efficient systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Simple electronic control systems are used widely in every aspect of life today. Motion sensors help conserve electricity by dimming lights when no activity is detected for a long period of time. Sensors on garages are programmed to turn on high-intensity lights when movement is detected. Water detection systems are used in household piping to detect leaks before permanent structural damage is caused. Most of the systems used in the real world today have one thing in common – they were designed by engineers.

The process of designing and constructing an electronic system is a task far beyond the capabilities of most people. Engineers generally start their designs by coming up with a list of specifications that their system must meet. A list of requirements for the system are drawn up. For example, with a garage light system, a system might be required to detect motion from a range of 20 feet. Also, the system may be required to only turn lights on in a dark environment, and to have the detection range be adjustable by the user.

Creating a list of requirements for a system is probably not beyond the ability of most average users. However, after requirements are drawn up, a schematic must be created that details how to assemble a system from components. Possible components include resistors, capacitors, microcontrollers, and FPGAs. The average user is incapable of evaluating components for use in a system, and certainly cannot create a complete working schematic from scratch. Even given a schematic, most users would not be capable of constructing a system Construction involves assembling components at a small level using wires and a breadboard or PCB. People unfamiliar with the notation of electronic schematics might get confused with symbols, pin numberings, and labels. In addition, constructing many systems requires the programming of a microcontroller or FPGA. The average system user has probably never written a computer program in his life, let alone an embedded program. Thus, the design and construction of electronic systems from components is beyond the ability of most average users.

Alternatives exist to assembling systems from individual electronic components. A whole class of designs known as off-the-shelf systems exists. An off-the-shelf system is pre-designed and pre-constructed to meet common needs of users. These systems do not require any design or construction from the user, and can generally be setup without the need for any technical knowledge. For example, Google [6] sells an off-the-shelf system for intranet searching known as the Google Search Appliance. This system allows the searching and archiving of data without any technical knowledge from the user. The only user requirement is that they read a manual on how to operate the system.

Off-the-shelf systems exist for a wide variety of tasks today. Motion detection, water sensing, and light sensing systems can all be found as off-the-shelf systems. For common tasks, an off-the-shelf system is likely to exist since manufacturers seek to build high-volume systems to increase profit. For obscure tasks, however, users may have a hard time finding an off-the-shelf solution.

Consider the following system: a user wants to automatically turn on a light outside his house only when it is raining and dark outside so that passing motorists can gain greater visibility. Off-the-shelf systems exist for turning lights on when water is detected, and for turning lights on in darkness. However, it is unlikely that anyone has designed an off-the-shelf system that illuminates when both water and darkness is detected. Ideally the user would like to take the two separate systems and somehow combine them, however there would likely not be a way to do this without modifying the system component internals. This is obviously beyond the ability of most users.

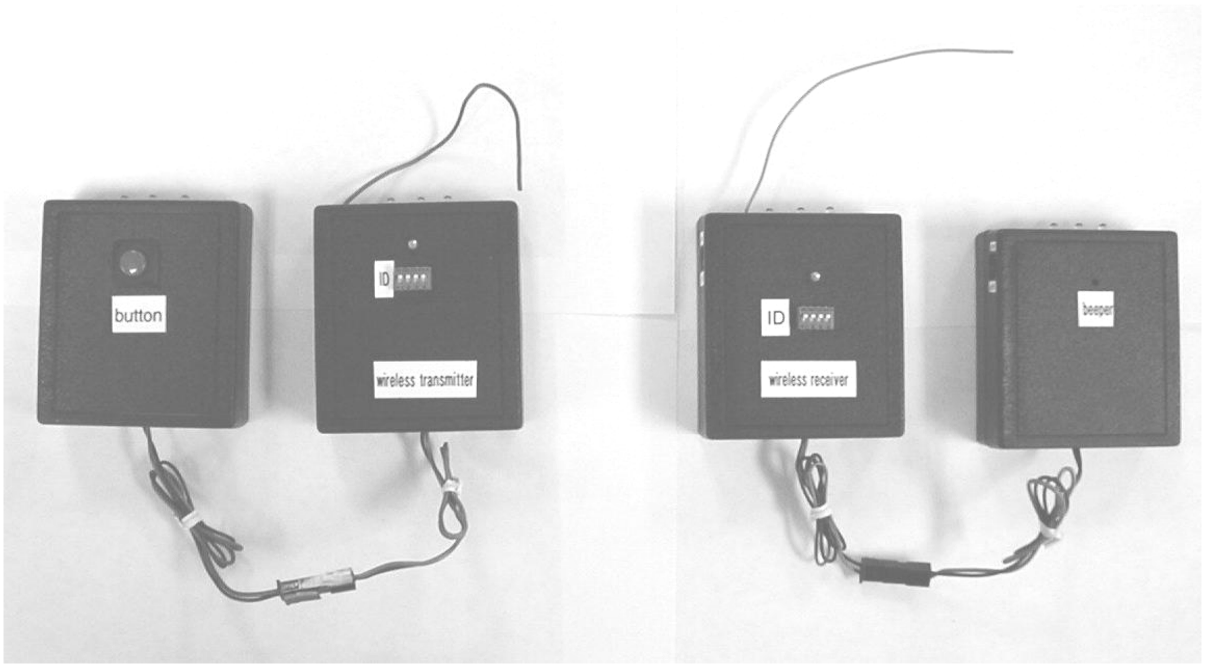
If a user wanted to build this rain/darkness detector system then off-the-shelf systems likely would not meet requirements. The easiest solution for the user would be to simply hire an experienced engineer to build the system for him. While the engineer would defiantly produce a working system, the monetary cost demanded would likely be high. It would likely take an engineer at least several hours to design and construct such a system, with a total cost of hundreds of dollars.

There is a third option for ordinary users rather than buying an off-the-shelf system or hiring an engineer to design a system. Devices exist which attempt to simplify the design process for ordinary users. These devices are typically modular and easy to understand, and can be combined to build custom systems that cannot be found off-the-shelf. In addition, the cost of these devices are typically far less than the cost of hiring an experienced engineer. The goals of these devices are generally to allow average users to have some degree of customizability in designing their own custom systems.

eBlocks [3] are single-function blocks that physically connect to create simple useful systems. Each individual block is powered by a PIC16F628 [16], communicating serially with other blocks using a two-wire connection. The eBlock communication protocol is binary: blocks communicate by sending “YES” and “NO” packets to each other. Figure 1 shows a wireless notification system constructed using eBlocks.

The eBlock library consists of mainly of fine-grained single function blocks that can be classified into three groups. Input blocks observe their environment, sending “YES” packets to respond to certain stimulus. Examples include light sensors, motion sensors, sound sensors, and push buttons. Output blocks produce an observable effect when “YES” packets are received, such as LEDs, buzzers, and electric relays. The third type of blocks are intermediate blocks that perform operations on inputs and produce outputs. The “OR” block which takes the logical “OR” of two inputs as its output is one example of an intermediate block. Another example is the wireless receiver and transmitter blocks that take an input and then communicate wirelessly to produce an output from a remote location.

Figure 1 – Wireless eBlock Notification System



The target users of eBlocks are those without electronics experience who want to build simple customizable systems, including but not limited to children. Normally, a user unfamiliar with electronics design would not be able to construct a system such as an alarm that triggers wirelessly when light is present; they would be forced to get by with an off the shelf solution that might not meet all the specifications, or hire an engineer to get the job done, which would be a costly proposition. However, by using eBlocks, even the most inexperienced users can build a complex system such as sensor networks [2] just by connecting component blocks together.

Several commercial and academic projects also have goals of allowing ordinary people to construct simple systems. Lego Mindstorms [12], based on MIT Bricks [13], allow the construction of mobile devices using LEGO blocks. LEGOs can be snapped together to build mobile vehicles and robots. A programmable Mindstorm block can then be added to control movement. Additional input LEGOs exist such as lights sensors and touch sensors. Using LEGO’s programming tool, LEGO robots can be programmed to perform interesting tasks. For example, a robot can be made to navigate a maze by responding to feedback on its motion sensor. Another possibility is a robot that greets people with they arrive home by detection of light. In general, Mindstorms are useful for users who wish to design robotics systems without first acquiring a vast amount of technical knowledge.

MIT Crickets [14] are miniature computers that communicate through infrared to build systems. Crickets such as sensors and displays can be programmed using a subset of the LOGO programming language [11]. MIT Crickets have been expanded into the commercial Handy Cricket project [7]. Handy Crickets allow interfacing between sensors, displays, and even other crickets by writing LOGO programs. Other common cricket peripherals include servo controllers, DC motor controllers, and electric relays.

One tradeoff made in the design process of eBlocks and Crickets was programmability. The designers of Crickets chose to make their devices programmable with LOGO. While this creates more options for interfacing and allows a greater degree of flexibility in designing systems, a user must write a program to achieve any task. On the other hand, eBlocks do not require any programming. The downside to eBlocks is that blocks can only interface together between other blocks. If some complex logic function of inputs is wanted, that function must be implemented using only eBlocks, while Crickets are capable of implementing the function entirely in software. Thus, Crickets sacrifice usability by novice users in return for greater usability by programmers and utility compared to eBlocks.

Logiblocs [10] are miniature blocks that perform logic functions such as AND and OR. Each block represents a different logic function, and blocks can be snapped together to build systems. Input blocks such as buttons provide input to the system, feeding output blocks like buzzers. An important distinction between Logiblocs and eBlocks is that eBlocks communicate by sending packets along a serial connection, while Logiblocs snap together to create an electrical circuit. Logiblocs can be thought of as friendly interfaces to logic gates. Blocks representing gate functions snap together to build an entire circuit. When the circuit is complete, current flows through the design similar to a normal circuit, only at a much larger scale. While most novice users would not be able to interface CMOS logic gates such as 74HC08 ICs [21], Logiblocs basically expand the form factor of these ICs and simplify interfacing to allow for ease of use.

Other projects target more experienced users who have medium to advanced knowledge of embedded system design. X10 [22] produces a line of home automation devices that communicate with a common protocol over power lines. X10 devices can be easily networked together to create custom systems such as security monitors. X10’s target market is users and businesses who want to implement customizable low-cost security and home automation systems. One system commonly sold by X10 is a video monitoring system. This system allows users to remotely monitor and record video from cameras with only minimal interfacing work. X10’s home automation systems allow homeowners to automate tasks such as turning lights on and off, and the control of multiple electronic devices using a single wireless remote. Overall, X10 systems are very useful for control systems involved in the house.

Other solutions, such as Phidgets [20], allow engineers and programmers to create complex designs using a computer as the central controlling device. Various single-function Phidgets connect to the computer via USB, and are programmed using Java, C++, or Visual Basic APIs. Many different tasks can be programmed by users, allowing for almost any interaction possible. Phidgets are very useful for experienced software programmers who have little experience with electronics. These programmers would probably be unable to design and interface components with a schematic. When building the system is reduced to simply writing interaction software, however, a programmer without any background in hardware design will thrive. In some sense, Phidgets allow all software programmers to be hardware programmers as well.

One common characteristic of the systems mentioned is that they are all self-contained. There currently exists no standard way for Phidgets to interact with Logiblocs for example or for any of the design systems to interact with the others. All of these design systems have their advantages and disadvantages. Some are easier to use than others, and some trade ease of use for programmability.

In the past, eBlock systems have been entirely self-contained. That is, eBlocks only communicate with other eBlocks inside systems. While eBlocks exist for many purposes, it is often necessary to build systems that interface with other networks. For example, if eBlocks could communicate with the Internet then many possibilities would be created for new systems. In addition, software tools can be used to increase the productivity of eBlock designers. Simulation tools allow users to test their systems before actually constructing them. Synthesis tools allow users to easily create systems in software that can be transferred to a real-life system.

In this paper we present several eBlocks used for interfacing to other devices. Our interfacing eBlocks allow for telephone, Internet, and Palm PDA connections. Also, we present two software tools for improving eBlock design and testing. Our eBlock simulator allows users to easily construct and test systems before construction. The eBlock synthesis tool allows the replacement of multiple intermediate blocks with a single powerful programmable block. Finally, we present a sample system that uses our new creations in order to demonstrate the usefulness of eBlock interfaces and software tools.

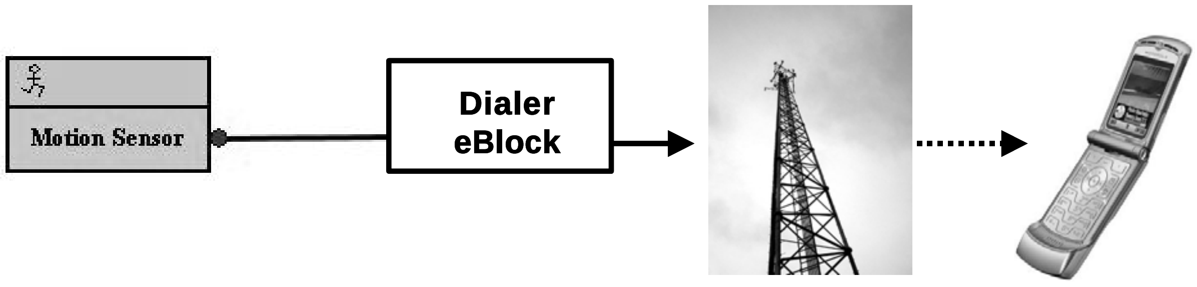
Chapter 2

eBlock External Interfacing

2.1 Overview

eBlock systems are entirely self-contained – a chain of eBlocks produce output that is processed by other eBlocks, until a final output is produced for the user. However, eBlocks are not ideal for certain operations. Building a long-range network of eBlocks across the world for communication would be expensive and foolish. Such a network already exists in The Internet, and if eBlocks could tap into that network then the cost would not have to be duplicated. Similarly, if eBlocks could interface with the telephone network, long range wired or wireless communication could be implemented with eBlocks without obtaining a special FCC permit. As eBlock designers, we follow the belief that tapping into a vast resource such as The Internet is a far better idea than trying to reimplement it using just eBlocks. In order to expand the possibilities with eBlocks, we designed three external interfaces: Telephone, Internet, and Palm PDA. These interfaces

Figure 2 - Phone Dialer eBlock System



greatly expand the possibilities for eBlocks by allowing numerous new systems to be created.

2.2 Phone Dialer

One area that eBlocks were previously limited in was remote communication. At the very short range, an LED is all that is required to notify a user of an event. At the range across a single small room, a beeper could get the job done. For communication across rooms or across buildings, the wireless transmitter and receiver can be employed. However, for distances across cities, states, or even countries, eBlocks had no solution. With this in mind, we set out to build an eBlock capable of communicating over any distance.

By tapping into the telephone network, eBlocks gain the ability to communicate across any distance. Since mobile phones are also accessible, eBlocks gain long-range telephone communication ability by using the phone network. Achieving such a long range of communication could not reasonably be achieved by any other means than using the existing telephone network. It is not realistic for a project such as eBlocks to be able to fund a giant telecommunications network or to build a satellite network.

Figure 2 shows the idea behind the phone dialer eBlock. A single eBlock input is provided, along with a telephone line output. Once powered on, the user inputs on the keypad to specify a phone number. This represents the number that the dialer will call. Users can enter a landline number to call a fixed destination, or a mobile number to reach someone anywhere. While NO packets are received, the dialer lies dormant with the phone line still closed. Once a YES packet is received, the dialer opens the phone line and dials the specified number. After a delay of about thirty seconds, the dialer hangs up. The remote user can identify the dialer calling by the remote identification of the dialing line.

The dialer is intended to notify users of events remotely, over extreme distances. Using a mobile phone, a user can receive an eBlock alert from across the world. This is useful for applications such as notification of intruders. If on vacation, a motion sensor can be setup in a room connected to the phone eBlock. If an intruder is detected, the house owner will get a remote alert and perhaps call his neighbor to check the premises. Another possible application is a remote doorbell. In a large noisy house, a doorbell may not be audible in a distant room or in the backyard. By using the dialer eBlock, it is possible to dial a cordless phone placed anywhere in the house when someone comes to the door.

Internally, the dialer uses a PIC16F628 to interface with eBlock systems. A separate 8051 microcontroller [9] is used to scan phone number input from the keypad and output to the LCD. The MT8880C IC [17] is used to create DTMF tones to be outputted to the phone line. These tones correspond to the 0-9 tones on a telephone. A relay is used to control the telephone line and only take the phone off the hook for a short while after a YES packet is received. A single PIC16F628 would be inadequate due to the lack of available I/O pins to drive all the peripherals necessary. An LCD, Keypad, and dialer IC must all be interfaced.

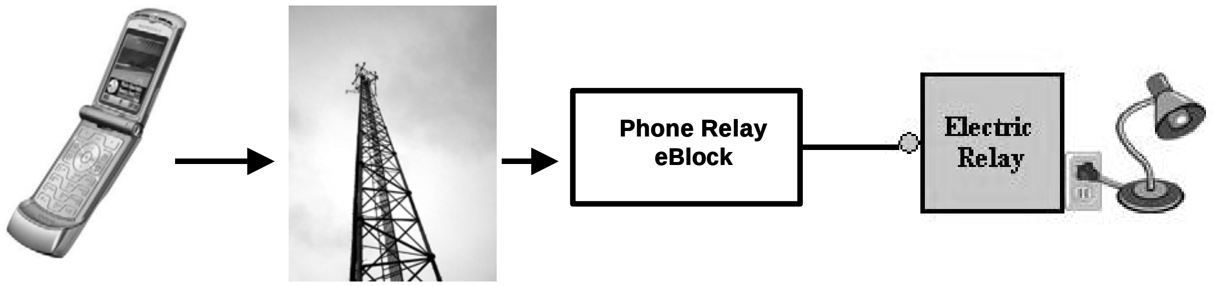
Since there are many high-power components in the telephone eBlock such as the LCD and the 8051 microcontroller, the power source for the dialer is an AC adapter. A tradeoff exists between the user interface and the power consumption of the system. Rather than including an LCD, we could have implemented the system without any user interface. We would have to assume that the user would enter the phone number correctly without any feedback on the entries. The tradeoff we chose allows for user friendliness at the cost of power consumption. Since we are forced to use a AC adapter, it also becomes a tradeoff in size and portability of the design. However, the downside of requiring a power socket is not severe. In general, wherever a phone line exists, a power socket is nearby. Since the phone dialer requires a telephone line to dial out, using a power socket is not much of a limitation at all.

2.3 Phone Relay

The phone dialer takes input from an eBlock system and remotely alerts the user of events. However, there is also the need for a system that acts in reverse. The phone relay takes input from a user over the telephone and remotely delivers the input to an eBlock system on the other end of the line. Figure 3 shows an example system using the phone relay.

The eBlock phone relay is intended to be set up as an input to a system such as one that controls lights. When the user is away, he can dial in to set the input to the system. After dialing, the user is given a tone that alerts him to enter a password. After

Figure 3 - Phone Relay eBlock System



the correct password is entered, a “0” can be pressed if he wants to send a “NO” packet to the system, or a “1” can be pressed if a “YES” packet is to be delivered. Depending on the user input, either a “YES” or “NO” packet will be outputted continuously at the standard rate of 2 seconds per output until the user calls back and supplies a different input.

The phone relay is useful for systems such as remote lighting for security purposes while on vacation. About once per day, the user can call in to the system to turn several lights on in his house. This would deter would-be thieves who might decide to rob the house if they notice no activity for several days. Another possibility for the phone relay is a remote garage door opener. If a user is away on vacation for a long period of time with a pet secured in the garage, the door can be opened remotely to get the pet some fresh air. The garage can then be closed again after a period of time for security.

The eBlock relay consists of a standard phone relay [19] connected to an eBlock interface. The phone relay handles the telephone side of the system, answering incoming calls and waiting the password to be inputted. Once the user requests a “YES” packet, the relay is triggered on. The eBlock interface consists of the standard PIC microcontroller taking input from the relay. The microcontroller detects when the relay is turned on and outputs “YES” packets to the eBlock system when necessary.

An interesting application of the phone relay and dialer blocks is the self dialer. By hooking up a phone relay to a dialer, a user can call the relay to turn the dialer on. If the dialer is programmed with the phone number of the person dialing then he will soon receive a call back! While impractical for use in actual systems, self-dialing is interesting nonetheless.

2.4 Palm Interface

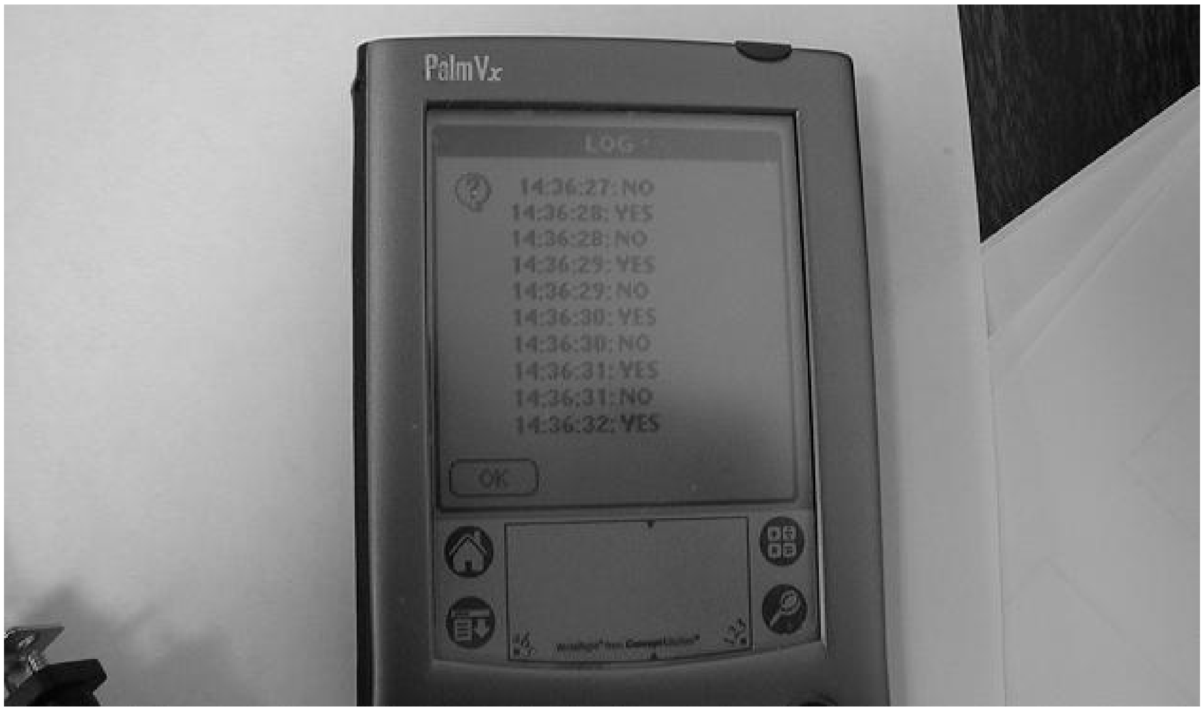
Many eBlock applications require the monitoring and logging of input and later drawing conclusions from the data in aggregate. An example would be the hallway logger system mentioned earlier or a freeway monitoring system. Road builders need to get justification to build new roads by monitoring and recording traffic patterns. Obviously a system that produces a one-time output such as a beep every time a car goes by would not be adequate. The data must be recorded automatically for statistical analysis later.

Designing a new eBlock specifically for monitoring and logging would be unnecessarily cumbersome and complex. Ideally, the block would have to have an LCD to display the incoming data in real time. Also there would have to be some kind of system of storing data files in memory. Finally, data files would have to be downloadable to a PC for analysis. This would require a serial port on the block and the necessary software written for the PC.

Because of the enormity of the task of building a stand-along logger, we decided to use a Palm Pilot to perform the task. In fact, a Palm Pilot the perfect device to perform eBlock logging. Palms are already very capable of receiving serial data from an eBlock through their serial port. The Palm’s LCD allows for display of data in real time. Data files can easily be saved to the Palm’s storage memory. For copying data to the PC, a “hotsync” application already exists for Palms. This can be send any data file on the Palm, such as the recorded traffic data files, back to the PC for storage and analysis.

We wrote a logging application using the serial port of the Palm to determine when “YES” and “NO” packets are received. When the application is started, it begins monitoring the serial port for traffic until stopped. When the application is exited, the time and type of each packet received is displayed. This data is then saved to a database to be uploaded to a PC and analyzed later. Figure 4 shows the output of the Palm logger for a sample application. We initially considered recording every packet received by the Palm, not only limited to changes in packet streams. However, during long periods of inactivity, this would flood the data file with a large number of “NO” packets. Since eBlocks are programmed to send packets every two seconds at maximum, one hour of logging all packets would equate to 1800 entries in the data file. In such a large data file, meaningful data can be lost as noise. However, if users demand the ability to record all incoming packets, the Palm logging program can be trivially modified to perform the task.

Figure 4 - Palm Logger



```
14:36:27: NO
14:36:28: YES
14:36:28: NO
14:36:29: YES
14:36:29: NO
14:36:30: YES
14:36:30: NO
14:36:31: YES
14:36:31: NO
14:36:32: YES
```

Ideally, eBlocks should be able to directly to the Palm. However, since the Palm has an RS232 serial port and eBlocks use TTL logic, the Palm cannot communicate directly with eBlocks. For the translation between TTL and RS232, a converter block is used. This block runs off a MAX232 IC [15], with an eBlock input and a serial output designed to connect to the Palm. A Palm output is also supplied to take input from the Palm and output to an eBlock system for future applications. Power is supplied to the converter block through a 9V battery. This converter block is necessary whenever interfacing a Palm to an eBlock. Thus, we have dubbed this converter the “Palm Converter Block”. It would likely ship with the Palm logging software in any future commercial distribution of eBlocks.

While the only application that currently exists for the Palm / eBlock interface is the logger, additional applications can be produced in the future. One possibility with the

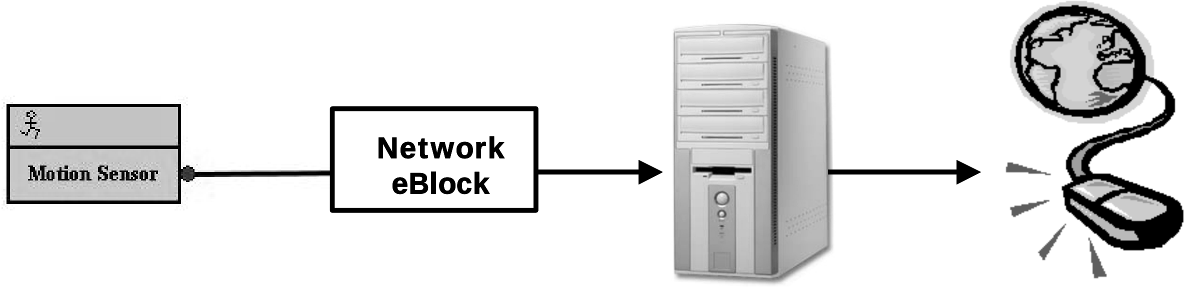
Palm is eBlock diagnostics. By adding a special port to communicate with the Palm, eBlocks can report information such as battery life and error status. This could be used for the debugging of eBlocks assumed to be broken. Another exciting possibility is using the Palm for providing updated firmware to eBlocks. With eBlocks based on integer packets expected to be released soon, it could be useful to upgrade legacy blocks from binary to integer. If the block firmware was written to include a routine for erasing and reprogramming, Palms could be used to stream firmware updates to blocks in the filed.

Many other possibilities open up for the Palm / eBlock interface with Palm to eBlock communication. One possible application is a timed signal generator. The user could specify a given time, and at that time a "YES" packet would be sent to the eBlock system. Many times during the day could be specified to toggle light switches on and off while on vacation. This would be a clear improvement to the phone relay-based light toggle system mentioned above. Another possibility would be using the Palm as an improved timer eBlock. Users could specify a delay time down to millisecond precision up to a very large timer limit rather than the 10 second, second-incremented timer block that currently exists. The possibilities for the Palm / eBlock interface are endless.

2.5 Network eBlock

For providing input to a single user, a notification system using the eBlock phone dialer is adequate. For notification of a large number of users, the phone dialer fails however. For example, suppose an eBlock system is setup to monitor motion in a hallway for security purposes. If motion is detected after closing then all security guards should be notified. Using phone dialers would require a splitter and a separate dialer for every

Figure 5 - Network eBlock System



security guard. Each dialer would require its own phone line and would have to be individually programmed with a different phone number. Obviously, this solution is not feasible.

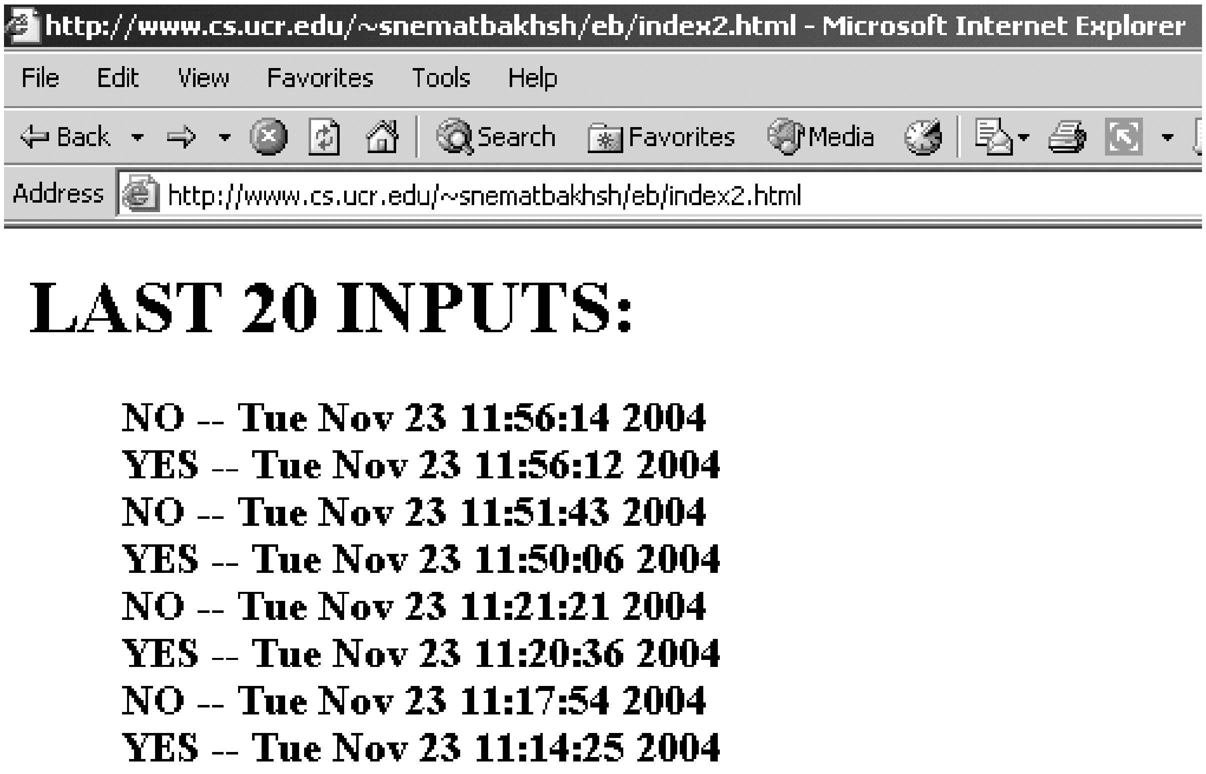
To solve the problem of multiple user notification, we created the network eBlock. This eBlock takes input from an eBlock system, encodes it into a packet, and sends the packet over the Internet to a specified computer. Figure 5 shows a sample system that uses the network eBlock. Whenever motion is detected in the hallway, a web page is updated with time information about the intrusion. This page can be monitored from any number of locations, providing multi-user notification.

When considering how to implement the network eBlock, we were faced with a tradeoff. One possibility would be to use TCP networking to send packets. Advantages of TCP include packet ordering and resending of dropped packets. However, the TCP protocol requires more advanced hardware to run. The alternative is to use UDP, which does not guarantee in-order packet reception or any packet reception at all. UDP simply sends the packet and hopes it is received, if not then no additional corrective action is taken. Since the use of TCP would incur additional NRE and unit cost expenses, we chose to use UDP. In doing so we sacrificed some system reliability. However, UDP is adequate for most systems that aren’t mission critical, which includes virtually all uses of eBlocks.

The network eBlock constantly executes its code to probe status of the eBlock based on the last eBlock packet received. A UDP packet is sent from the network eBlock at a rate of approximately 100 per second. Thus, it is generally not a problem when a packet is lost due to the UDP protocol choice since another packet will be incoming just 10 milliseconds later. How this data is handled is entirely up to the user. Our monitoring application looks only for changes in input, and then records the date of the input change, posting it to a web page. Figure 6 shows the output of this application. Thus, if only “NO” packets are received for a long period of time, the output will not be flooded with useless information.

It is possible that a user will want to display all packets received or handle input in some other way. For example, rather than displaying a message on every change in input, it is possible that users would want to only observe the block state every two seconds and record the data at those times, ignoring all data between. Another possibility is the emailing of users once a “YES” packet is received to alert them of problems. Such an application would be useful for a water sensor system to check for water heater leaks. While our network eBlock does not currently include applications to perform these tasks, programmers can easily customize use of the network eBlock through our API. Our API contains functions to receive packets from the network eBlock so that anyone with C programming experience can create whatever application is required. By simply writing a function call, a packet can be received from the network eBlock. No internal knowledge

Figure 6 - Network eBlock Web Page Output


http://www.cs.ucr.edu/~snematbakhsh/eb/index2.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.cs.ucr.edu/~snematbakhsh/eb/index2.html

LAST 20 INPUTS:

```
NO -- Tue Nov 23 11:56:14 2004
YES -- Tue Nov 23 11:56:12 2004
NO -- Tue Nov 23 11:51:43 2004
YES -- Tue Nov 23 11:50:06 2004
NO -- Tue Nov 23 11:21:21 2004
YES -- Tue Nov 23 11:20:36 2004
NO -- Tue Nov 23 11:17:54 2004
YES -- Tue Nov 23 11:14:25 2004
```

of the network eBlock is necessary to write a useful application. Thus, using our API is very simple and straightforward even for novice programmers.

To use the network eBlock, an IP address must first be entered on the keypad. This represents the IP of the listening computer. An LCD displays the IP data as it is entered. Next, the network gateway must be supplied for packet routing. Both the keypad and the LCD are controlled by an 8051 microcontroller, and the eBlock interface is controlled by a standard PIC microcontroller. When input is received, packets are sent to the target computer using a CS8900A [1] network controller, driven by the 8051.

Currently the network eBlock only provides for communication from an eBlock system to a computer network. A future project would be to implement communication in the reverse direction, from a computer network to an eBlock system. This would allow for some interesting systems such as allowing users to control real-world events using a web page. Another interesting application would be to implement communication between eBlock systems across a network. This way, an eBlock system could communicate with another system across the globe.

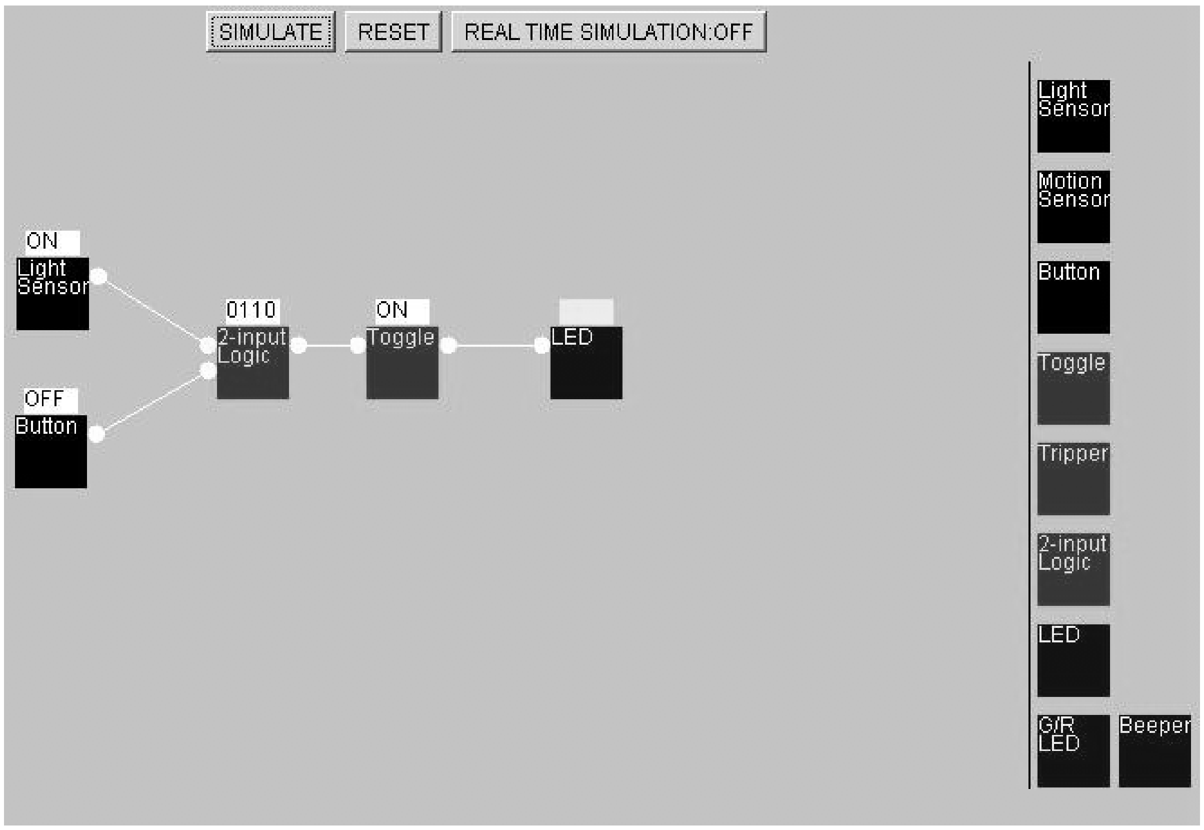
Chapter 3

eBlock Simulation and Synthesis

3.1 Overview

eBlocks were intended to be snapped together at the physical level without any need for programming or software tools of any kind. However, software tools can be used to aid in the construction of systems. Simulation can be used to layout, test, and debug eBlock systems before they are physically constructed. With a well-designed interface, even users with little computer experience can can use a simulator to aid in development. For advanced users, synthesis allows the creation of custom eBlocks. Multiple single-function blocks can be replaced with a single custom synthesized block. This synthesized block can then be burned to a chip and attached to the programmable eBlock. As an additional feature, synthesis allows the simple creation of systems using “virtual blocks” which exist entirely in software, without a physical block counterpart. Since were are dealing strictly with software, there is no need to limit our eBlock library size.

Figure 7 - eBlock Simulator



3.2 eBlock Simulation

We developed a Java applet-based drag-and-drop eBlock simulator to aid in system design. The simulator features a menu of available real-world eBlocks such as buttons, 2-input logic, and LED outputs. Virtual systems are built by dragging blocks from the menu and placing them in the workspace. Connections between blocks are made by dragging between targets on blocks to create wire connections. Figure 7 shows a sample layout of an eBlock system using the simulator. In this example we have a light sensor and a button input connected to a 2-input logic block. The 2-input logic is specified to output “YES” only when either the light sensor or the button output “YES”, but not both (hence the vector “0110”). This output is then fed to a toggle block and outputted to an LED.

Once a complete system is laid out, the user can begin to setup the system state for simulation. Users can click on input blocks to set their input states to YES or NO. For example, clicking on a button block and setting its value to YES would be the equivalent of pushing the button on the block. Certain blocks like the 2-input logic block require user customization to specify their function. This customization is also performed by clicking on the block. Above the block appears its state so users can know what customization they have set.

Simulation is performed by clicking on the simulate button. One iteration of simulation occurs each time the simulate button is clicked. For each click, each of the input blocks sends exactly one packet with the value of its current state. These packets propagate across the entire system, producing a final stable state. There also exists a real-time simulation mode that can be toggled on with the click of a button. When real-time simulation is enabled, simulation is repeated every second. The simulator functions as if the simulate button was pushed every second until real-time simulation is toggled off. When simulation is completed, the results are displayed on the output nodes. For example, an LED output node will light up, and a buzzer output node will produce an audible alert.

Our simulator aims to allow for accurate designs at the system level. Systems laid out using the simulator are intended to behave in the same manner as similar systems created in hardware. This high-level simulation approach is used because our simulator is

Table 1 – Simulation Algorithm

```
Simulate(table T[n])

1)   Let Eval[n] = {-1 … -1}
2)   Let update = 1, proc=1
3)   while update ==1 do
4)       update = 0
5)       for i = 1 to n do
6)           Let j_1 … j_k = inputs(T[i])
7)           if Eval[j_1] != -1 and  Eval[j_2] != -1 … and Eval[j_k] != -1
8)               Eval[i] = evaluate(T[i])
9)               Ordered[proc]=T[i]
10)              proc++
11)              update = 1
12)  CopyOutput(T,Eval)
```

intended for eBlock users. From the user perspective, individual eBlocks are simply black boxes with inputs, outputs, and perhaps a switch or dial. Thus, creating a cycle-accurate simulation of the eBlock architecture would not be useful for users. Users are not concerned with then underlying architecture of eBlocks, they simply interact with the user interface. The eBlock architecture could change dramatically, replacing microcontrollers with FPGAs for example, and users would not notice since they only interact with the outside interface. This external interface is what is provided in our simulator.

Internally, a table of blocks represents the eBlock system. As the user lays out blocks, table slots are allocated and filled in with the function of the block the user chooses. As wires are drawn to connect blocks, the wires are recorded in the table. The table is constantly updated whenever a wire is added or deleted to completely reflect the current layout of the system.

Our simulation algorithm is shown in Table 1. The *Simulate* algorithm takes the table representation of our system. The *Eval* array is used to hold the output of each block in the simulator, initialized to -1 to represent that the output has yet to be defined. The algorithm begins by trying to find an entry in the table whose inputs are all defined. In the first pass through the table, this will only be the input blocks such as buttons that do not have any inputs. The *evaluate* function evaluates the output of a block in the table based on its inputs and the block type. For example, if the block were an 2-input logic block, the output would be based on the two inputs and the internal vector representing the block customization. For input blocks like buttons, the evaluation is based entirely on the internal customization of YES / NO and not on any inputs.

After the first pass through the table, repeated passes are made, provided that some block was evaluated in the previous pass. The *update* variable is used to keep track of updates to the *eval* table. If no updates are made on an entire pass then *update* will remain false and iteration will stop. At most, the number of passes through the table will be *n+1*, and each pass requires the checking of *n* table entries. Since the number of inputs to a block is bounded by a constant (currently 2 maximum inputs) the runtime of the function is O(n^2). While not ideal, the quadratic runtime will never cause problems due to the small size of eBlock designs. It is very rare to find an eBlock system with more than 20 blocks.

Note that every block in the table may not be evaluated. Certain blocks may be impossible to evaluate. For example, if an output block has no input then no evaluation will occur. After the *eval* table is completed, the *CopyOutput* function changes the status data of all blocks in the system to reflect the proper state based on simulation. For example, if an LED output was evaluated to 1 then it would light up in the simulator. Intermediate blocks like toggles display the current value they are outputting. A side effect of the *Simulate* algorithm is that an ordered table of blocks is created inside the *Ordered* array. This array contains blocks in the order they were evaluated, guaranteeing that the blocks can be evaluated in array order in a single pass. In addition, this array leaves out blocks missing inputs as mentioned above. The *Ordered* array will be a vital part of our *Synthesize* algorithm later.

When placing a wire, it is possible that a cycle could be created between several blocks. For example, the output of a toggle block could be fed to a 2-input logic block, and then fed back into the toggle. When building a cyclical system with actual eBlocks, the behavior will often be undefined. However, it is possible to build useful eBlock systems that contain cycles. For example, implementing an FSM+D with eBlocks would require bidirectional communication between the FSM and datapath units, and a cycle would have to exist somewhere. For simple control systems, however, there is generally not much of a need for cycles.

Our algorithm steps through the table of blocks, processing each block exactly once. The algorithm could easily be changed to evaluate a block multiple times, once whenever its input changed. This change would accommodate cyclical systems, however it leads to a problem. Malformed cyclical systems could cause logic race conditions, leading to an infinite loop. Therefore, in order to remove ambiguous behavior from our simulator, cyclical wires are disallowed. When a user places a wire in the design, a cycle check is performed. If a cycle is found then the placed wire is removed.

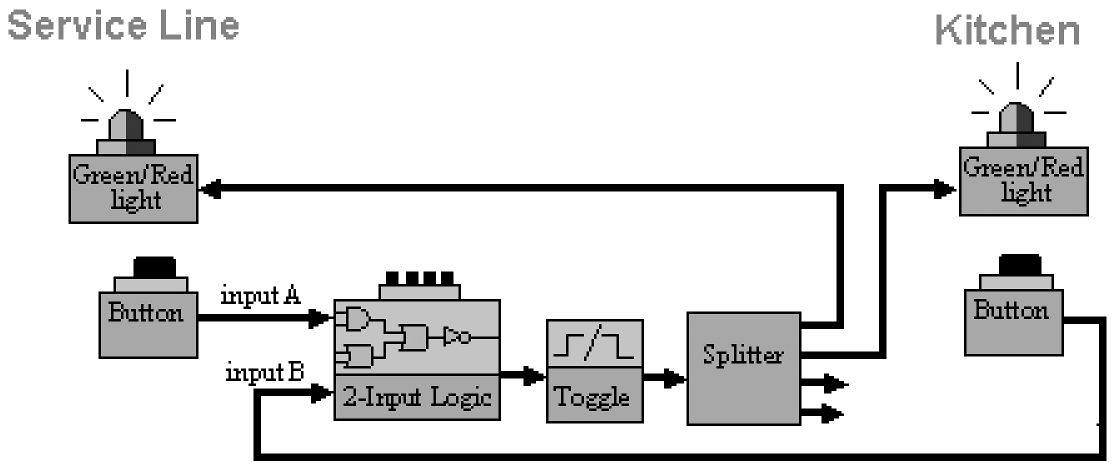
Since the simulator is a Java applet, any user with a Java-enabled web browser and Internet access can use it. The interface is intended to be usable by anyone with moderate computer literacy, not just engineers. The drag-and-drop placement of blocks and wires and the right-click deleting of items is intended to be intuitive to the average computer user.

There are many advantages to using our simulation tool. Our simulator allows users all over the world to construct eBlock systems without having access to physical blocks. Once eBlocks go into mass production, our simulator will allow users to get a sample of eBlock design without making a purchase. With our simulator, users can determine how eBlocks can meet their needs, and then purchase a set of blocks once they realize their usefulness.

Our simulator has already been an invaluable testing and feedback tool for eBlock designers. Many users have sent feedback about eBlocks without ever touching a physical block. As previously mentioned, a key design goal of eBlocks is ease of use. It is not acceptable to have blocks that are too complex to use for the average user. Our simulator allows designers to easily get feedback and make revisions on block designs. For example, the 2-input logic eBlock was revised after user feedback saying the block was difficult to understand.

Our eBlock simulator is a significant part of eBlock development. It is the basis for the current version of the simulator that is in use by over 500 people today.

Figure 8 - Cafeteria Alert System



3.3 eBlock Synthesis

Many eBlock systems require a large number of blocks. Figure 8 shows a cafeteria alert eBlock system. Building this system requires three intermediate computation blocks in between the inputs and outputs. If users could somehow create a multi-function customizable block to replace those three single function blocks, the number of blocks in the system and the power requirements would be reduced dramatically. With this idea in mind, we created the eBlock synthesis tool.

Our eBlock synthesis tool uses an interface identical to the eBlock simulator where blocks and wires are dragged and dropped onto the workspace. The menu of blocks for the synthesis tool is slightly different from the simulator however. Replacing the different types of input blocks such as buttons and sensors is a single input block. Input blocks placed in the circuit represent the external input points to the circuit. Any input can be fed into the block at this point. Similarly, outputs such as buzzers and LEDs are replaced with an output block representing an output point. Function blocks included

Table 2 - Synthesis Algorithm

```
GetOrdered(table T[n])

1)    Call Simulate(T)
2)    Keep Ordered array and output to C code

Synthesize (table Ordered[m])

1)    for i=1 to m do
2)        eval[i] = evaluate(Ordered[i])
3)    output = eval[m]
```

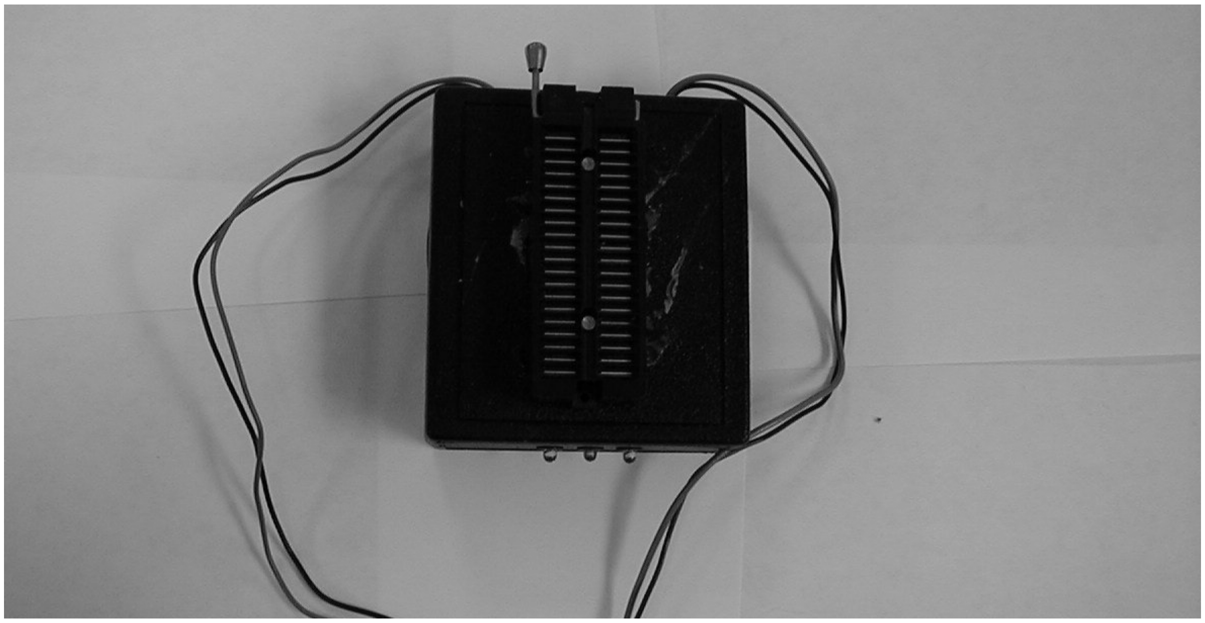
in the synthesis tool are complete eBlock functions like splitter, tripper, toggle, and 2-input logic.

Once the design is fully laid out with input points and output points, the synthesis button can be clicked for completion. Table 2 shows the synthesis algorithms used on a single input, single output design. First, the *Simulate* algorithm is executed to produce the ordered table of blocks, *Ordered*. This executes entirely in our Java applet. This ordered table is then translated to a C code representation and copied to the C output file.

Also copied to the C file is the code to perform the *Synthesize* algorithm. This code steps through the ordered array from front to back, recording the intermediate outputs along the way in the *eval* table. Once the final output node is encountered, that value is outputted. The *Synthesize* algorithm will be repeated once every time a new packet is received, or once per timeout period. Also included in the C code is serial communication code needed to implement the eBlock communication protocol. This code is for the receiving and sending of eBlock packets. The same code for *Synthesize* will be included in every C file generated by the eBlock synthesis tool. The only variation will be in the *Ordered* table that represents the user's design.

The runtime of *GetOrdered* is O(n^2) since it makes a call to the O(n^2) *Simulate* algorithm. The *Synthesize* algorithm, however, is only O(n) since we use the ordered table previously produced. This is an ideal partitioning of computational power. Since the Java code executes on a fast desktop machine, it can easily perform the O(n^2) algorithm in a reasonable time to produce the ordered array. The code for *Synthesize*, however, executes on a comparatively slow microcontroller. In addition, the microcontroller is intended to run off a battery. Thus, any additional execution time leads to additional power consumption and thus decreased battery life. This is why we chose to have the "front end" Java side of the synthesis perform the brunt of the labor, while the "back end" embedded side needs to only perform a simple linear table transversal.

Figure 9 - Programmable eBlock



Once our synthesis tool generates the code, the user can compile it using HI-TECH PICC [8]. The resulting binary can then be programmed onto a PIC16F628 microcontroller and downloaded onto the programmable eBlock, shown in Figure 9. This block consists of a single eBlock input, a single eBlock output, and a ZIF socket for easy placement of the programmed microcontroller. Blocks with multiple inputs and outputs also exist, however they require multiple microcontrollers due to the single UART per PIC chip. For these blocks, an additional ZIF socket exists for each set of inputs and outputs. The synthesis tool generates code for each of these microcontrollers, where one microcontroller performs all graph operations and the others are used strictly for I/O.

The programmable eBlock tool contains functional blocks that correspond to physical blocks that exist in the eBlock library. Toggles, trippers, and splitters are all physical blocks that are also included in the programmable library. However, since we are dealing with just a piece of software, there is no need for blocks in our synthesis menu to correspond to physical blocks in reality. Physical blocks are designed to be as simple as possible to be usable in more designs. This creates a small library that can be used to build many systems. With our synthesis tool, however, there is no need for this restriction. Placing blocks in a design is as simple as dragging and dropping from a software menu. Creating hundreds of new physical blocks would mean carrying around a suitcase full of blocks that are rarely used. In software, however, we can include any number of blocks, as long as our menu is organized and blocks are easy to locate for the user. We call these blocks that exist only in software “virtual blocks”.

Wanting to design virtual blocks for our synthesis tool, we created two blocks as a first step. The N-timer block is designed to output a YES packet only when YES packets have been received for N consecutive seconds. Until N seconds of only YES packets have been received, the output of the N-timer is NO. The value N can be custom set by the user by clicking above the block, with the value 1 through 59,999. This wide range allows the construction of long-range timing systems that would otherwise be impossible due to the limited range of the timer block. The other new block, the sequencer, is intended to match regular expressions. It observes the previous 8 inputs, and if they match the set pattern then a YES is outputted. If the pattern isn’t matched then NO is outputted. The user can set the 8-character pattern with mouse clicks. Patterns can consist of 1’s, 0’s, and *’s. 1’s represent YES packets received, 0’s represent NO packets, and *’s represent “don’t cares”. For example, a pattern of “******10” will output YES if the previously received packet is NO and the one before that was YES.

These two new blocks allow additional eBlock systems to be created. For example, the N-timer block allows the creation of long-range timing systems. The physical eBlock timer only allows the timing of systems up to 10 seconds. This limitation is due to the fact that it is difficult to specify large values on a physical block. For the physical timer block, a dial switch is used to specify the value of the timer. Specifying an N-digit timer value would involve setting N switches using this method, which would not be ideal. Another solution for physical blocks could be to use a keypad and an LCD for the timer value. However, the power consumption of LCDs is generally very high. Implementing an LCD on a physical block would mean a significant sacrifice in battery life. The best way to specify a large value is to use a PC with a keyboard and simply type the value in. Our simulator tool allows for this easy, simple input method.

Chapter 4

Sample Application: Parking Lot Monitor

4.1 Overview

eBlocks can be used to construct many useful electronic systems. In this section we look at using eBlocks to design a system for monitoring a parking lot. We present an eBlock system using only standard blocks, and then show how it can be improved using one of our eBlock external interfaces. Finally, we use our eBlock synthesis tool to further improve the system design. The end result is a system that is less complex and more powerful than the original system composed of only basic eBlocks.

4.2 Parking Lot Monitor

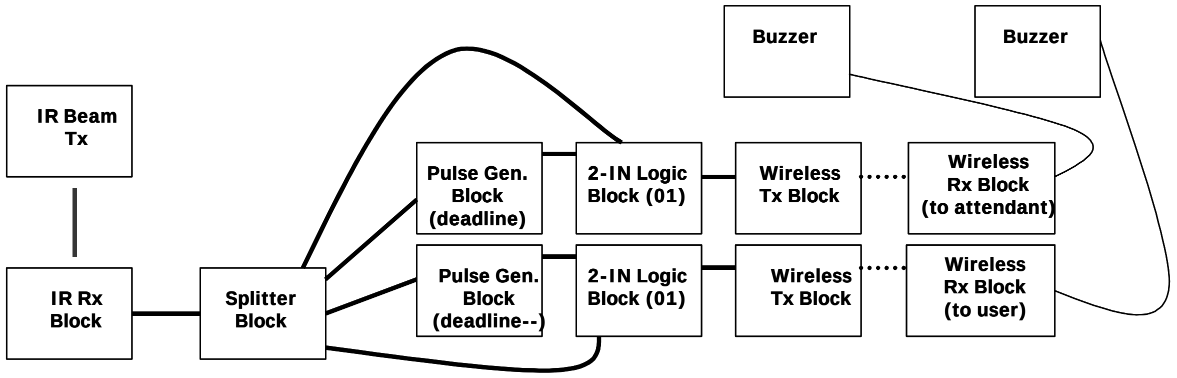
In many urban areas of the United States there exist time-limited meter–free parking spaces. For example, at UC Riverside there are several two-hour parking spaces adjacent to the Surge Building. While these spaces are convenient for users who make short trips, a problem arises in enforcement of the two-hour time limit since a parking attendant cannot possibly remember thousands of vehicles.

One solution commonly used to enforce the limit is chalk tire marking. When the attendant visits a given lot, she marks the tire of every vehicle in chalk. After a period of time the attendant returns, and any cars with marked tires are given tickets for exceeding the time limit. Several problems exist with chalk tire marking. First, if the period of time in between attendant visits is greater than the limit time of the parking space, people can get away with exceeding the limit. Also, chalk marks on tires can be removed through weather conditions like rain and snow.

We sought to design an electronic system to replace the existing method of chalk marking tires. Ideally, the system would be able to alert both the car driver and the parking lot attendant once time expired. Also, the system should measure the parked time exactly and produce an alert at the exact time the limit is exceeded. This would eliminate the possible gaps with a tire marking system. The result of such a system would be completely fair parking lot enforcement, and the saving of the attendant’s time.

An infrared or optical monitor would be placed on two polls on different sides of the space. When a car parks in the space, the breaking of the beam starts a timer. The car driver then picks up a receiver module and goes about his business. When the timer counts up to reach the limit time of the space minus a few minutes, a signal is sent to the driver’s receiver producing an alert. This warns the driver that he has nearly exceeded the limit time and must move his car soon or receive a ticket. Once the limit time is exceeded, a separate alarm labeled with the parking space number goes off with the

Figure 10 - eBlock Parking Monitor System



parking attendant. Since time has expired, the attendant can go and ticket the vehicle in the space.

4.2.1 Construction from Basic eBlocks

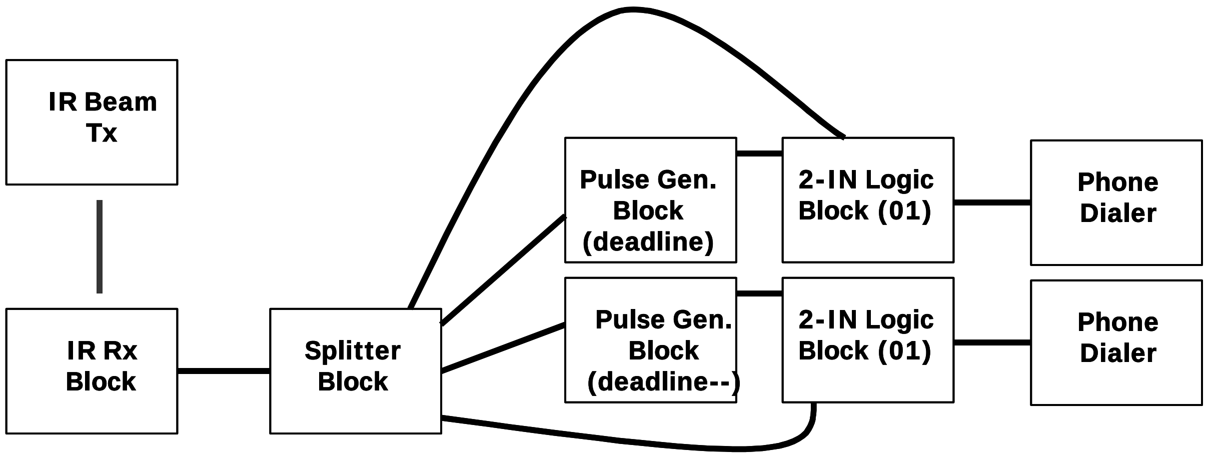
Figure 10 shows the design of our eBlock system. The infrared transmitter and receiver are placed across the parking space. When a car arrives, the beam outputs a YES packet, triggering the pulse generator block. The pulse generator blocks output NO until a YES packet arrives, at which time it outputs YES for a set amount of time (1 to 100 minutes) and then NO for a set amount of time (1 to 10 minutes) before it repeats. The YES duration gets set to the limit time of the space for the attendant output and to the limit time minus a few minutes for the user output. The NO duration is set to the desired duration time of the alert (almost always 1 minute).

In order to detect when the time limit has been exceeded, we need to check the state of the pulse generator and the infrared beam. If the infrared beam is broken (there is a car in the space) and the output of the pulse generator is NO (the car has been in the space for an excess amount of time) then an alert must be triggered. To check these conditions we use a 2-input logic block. The block is programmed to output YES only when the infrared input is YES and when the pulse generator output is NO. For all other combinations of inputs the output is NO. This output is then fed wirelessly to both the attendant and the car user. Several minutes before the time limit is exceeded, the user’s alarm gets triggered, warning him to move his car. Several minutes later, if the car is still present then the attendant alarm is triggered. While this system is more complex than typical eBlock systems, it is not unreasonable for an engineer with some eBlock experience to be able to complete design. Obviously it would be unreasonable to expect grade school children to design such a system.

4.2.2 Construction Using eBlock Dialer

Rather than using wireless transmitters and receivers connected to buzzers, we can use our eBlock phone dialer to produce the timeout alerts. Figure 11 shows our eBlock system using the phone dialer. When first parking their car, the user enters their mobile phone number to set one of the dialers. The other dialer maintains the phone number of the parking lot attendant. Several minutes before timeout, the first dialer calls the user’s mobile phone to give a warning. A few minutes later, the attendant receives a call to notify them of the infraction.

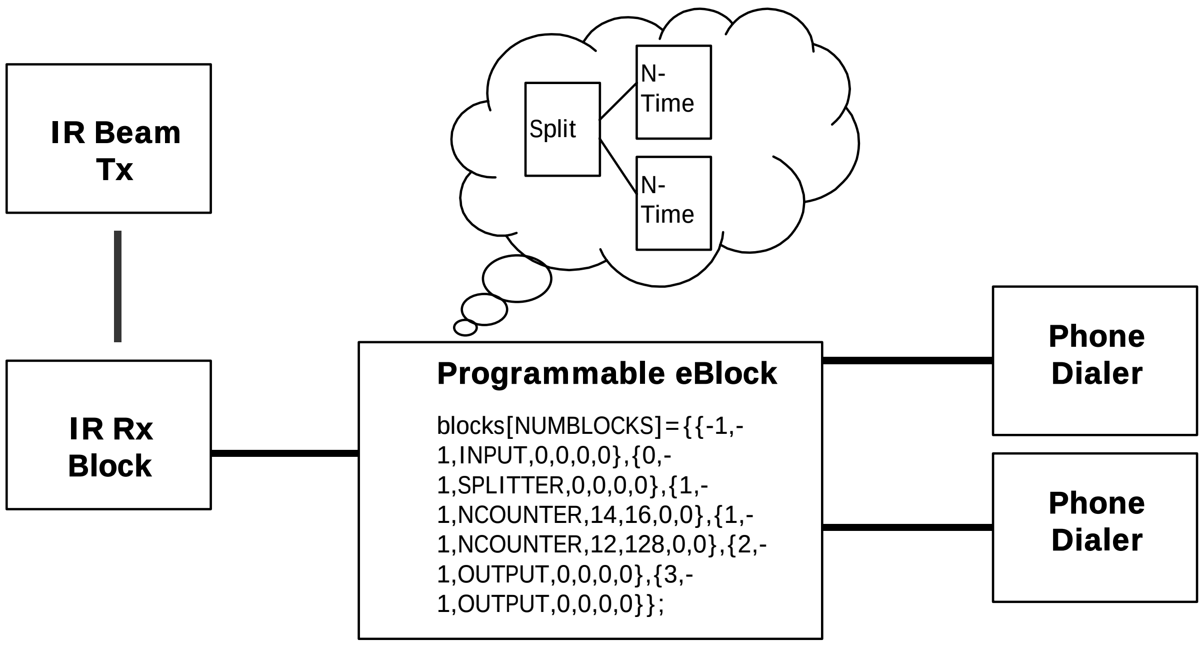
Figure 11 - eBlock Parking Monitor System with Dialer



In order to use the phone dialer version of the system, phone and power lines must be present near the site of the parking space. Thus, in some places this version will not be able to be deployed. However, the dialer version of the parking lot monitor system is superior to the version with only basic blocks. The biggest problem with the original system is that the range of the wireless eBlocks limits the range of notification. The wireless eBlocks aren't capable of sending a signal past a range of several hundred feet, so user notification becomes worthless at ranges longer than this.

The eBlock dialer solves this problem by dialing the user's mobile phone to send an alert. This means that the range of the system is limited only by the range of the mobile phone network. Users can travel almost anywhere and still receive their alerts. Obviously the user must have a mobile phone in order to utilize the dialer to receive an alert, however this will not be a problem for the majority of users. In addition, letting a mobile phone be the alert device means the user does not have to carry around a buzzer to receive an alert, which is a small edge in convenience.

Figure 12 - eBlock Parking Monitor System with Programmable Block



```
blocks[NUMBLOCKS]={{-1,-
1,INPUT,0,0,0,0},{0,-
1,SPLITTER,0,0,0,0},{1,-
1,NCOUNTER,14,16,0,0},{1,-
1,NCOUNTER,12,128,0,0},{2,-
1,OUTPUT,0,0,0,0},{3,-
1,OUTPUT,0,0,0,0}};
```

Another major advantage of the phone dialer version of the system is the block reduction it provides. Six wireless receiver, transmitter, and buzzer blocks are replaced with two dialer blocks. This leads to a system that is less complex and has better battery life. The wireless blocks in particular frequently require battery replacement. Replacing the battery-powered wireless blocks with the power line fueled dialer eBlock allows the system to run longer between battery replacements. Additionally, the dialer version would likely be cheaper since six total blocks are reduced to just two.

4.2.3 Construction Using Programmable eBlock

The timing portion of the original system is very complex and non-intuitive. A splitter, two pulse generators, and two 2-input logic blocks are used to produce timeouts when a car has been parked for too long. Even for experienced eBlock designers, it is difficult to determine the function of the system without close analysis. By using a programmable eBlock, we can replace the complex timing system with a single custom block. Figure 12 shows the parking lot monitor system using the dialer eBlock and the custom programmable eBlock. Our programmable block is set with a single splitter and two N-Counter virtual blocks. One N-Counter is set to the time limit minus a few minutes for user notification, and the other is set to the time limit for notification of the attendant. Each of these counters output to a phone dialer block to perform notification.

Comparing our original parking lot system to this new one, we see that the block count has been reduced from thirteen to five. This results in a dramatic reduction in power consumption. In addition, the construction of the final programmable block version is much simpler than the version using only basic blocks. It is easy to see what the function of the programmable version is since the design is much simpler. The fact that a custom block is used does not add much complexity at all. Any user can drag and drop the necessary blocks in software and then click a button to generate code. Clearly, this final version with the dialer and programmable block is superior to the previous constructions.

4.2.4 Construction Using Components

In addition to our different eBlock designs of the parking lot system, we also constructed a version of the system from electronic components. These components were purchased from the popular online electronics warehouse Digikey [4]. In addition, we were able to locate a schematic for the infrared portion of the system [5], which saved a large amount of time.

The total time necessary to design and construct the system from components was 8.5 hours. This is significantly more than the 1.5 hours to construct the original eBlock version. Rather than using wireless alerts, the component system uses a wired alarm, limiting the range of the system even more than the wireless eBlocks. While it is often necessary to construct systems from components in order to meet design specifications, in this case the eBlock version of the system was able to meet all necessary specs. Thus, for the parking lot monitor system, eBlocks are a superior choice to standard components.

Chapter 5

Conclusion

5.1 Summary

We created a set of new useful and innovative eBlocks to interface with outside systems. By constructing the phone dialer, the phone relay, the network eBlock, and the Palm PDA logger, we expanded the possibilities for eBlock systems enormously. In addition, we created software tools to help eBlock designers build their systems. Our simulator tool allows users to layout and test eBlock designs without constructing physical systems. Our tool is the basis for the current eBlock simulator that is used by over 500 people today. We expanded our eBlock simulator to perform synthesis, allowing users to layout designs in software and then convert them to hardware. With our programmable block, users can simplify designs by replacing multiple blocks with one, and use special virtual blocks to make system construction easier. Finally, we showed how our interfacing blocks and programmable blocks greatly improve upon a sample real-world system.

5.2 Future Directions

Much work can be done in expanding the eBlock synthesis tool. Currently, there is no optimization performed on eBlock designs created by the user. If optimizations were added to the tool, our generated code could be improved to be simpler and consume less power. While there are only two virtual blocks in our tool now, many more could be added. Research needs to be done to show what virtual blocks are useful, and then these blocks can be added to the virtual library. Finally, additional external interfaces for eBlocks can be created. One useful example would be a network block that communicates from the Internet to eBlocks. This would allow interesting applications, such as controlling an eBlock system from a web page.

References

[1] Cirrus Logic. http://www.cirrus.com

[2] S. Cotterell, K. Downey, F. Vahid. Applications and Experiments with eBlocks - Electronic Blocks for Basic Sensor-Based Systems. Sensor and Ad Hoc Communications and Networks (SECON), October 2004.

[3] S. Cotterell, F. Vahid, W. Najjar, H. Hsieh. First Results with eBlocks: Embedded Systems Building Blocks. CODES+ISSS Merged Conference, October 2003.

[4] Digikey Inc. http://www.digikey.com.

[5] Electronics Zone. http://www.electronic-circuits-diagrams.com.

[6] Google inc. http://www.google.com.

[7] Handy Cricket. http://handyboard.com.

[8] HI-TECH Software. http://www.htsoft.com.

[9] Intel Corp. http://www.intel.com.

[10] Logiblocs Ltd. http://www.logiblocs.com.

[11] LOGO Foundation. http://el.media.mit.edu/logo-foundation/logo/programming.html

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

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

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

[15] Maxim Semiconductors. http://www.maxim-ic.com

[16] Microchip Technology Inc. http://www.microchip.com.

[17] Mitel Networks. http://www.mitel.com.

[18] Palm Corp. http://www.palm.com.

[19] Quasar Electronics. http://www.quasarelectronics.com

[20] Phidgets Inc. http://www.phidgets.com.

[21] Philips Semiconductor. http://www.semiconductor.philips.com.

[22] X10 Corp.http://www.x10.com/home.html.