

A Logic Block Enabling Logic Configuration by Non-Experts in Sensor Networks

Susan Cotterell and Frank Vahid*
Department of Computer Science and Engineering
University of California, Riverside
{susanc, vahid}@cs.ucr.edu

*Also with the Center for Embedded Computer Systems at UC Irvine

ABSTRACT

Recent years have seen the evolution of networks of tiny low power computing blocks, known as sensor networks. In one class of sensor networks a non-expert user, who has little or no experience with electronics or programming, is required to select, connect and/or configure one or more blocks such that the blocks compute a particular Boolean logic function of sensor values. We describe a series of experiments showing that non-expert users have much difficulty with a block based on Boolean logic truth tables, and that a logic block having a sentence-like structure with some configurable switches yields a better success rate. We also show that adding color to a truth table improves results over a traditional truth table.

Author Keywords

Sensor networks, Boolean logic, embedded computing systems, truth table.

ACM Classification Keywords

H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION

The continued shrinking of computer chip size and cost has led to a class of computing known as sensor networks [1,7,15,21]. A sensor network is a computing network consisting of tens to thousands (or more) of tiny compute nodes. A node may range in size from perhaps a matchbox, to something not much larger than a large piece of dust as shown in Figure 1, and may cost just a few dollars to as little as just a few cents.

One particular evolving class of sensor networks is known as eBlocks [3], which we are developing at the University of (withheld for blind review) as a project of the U.S. National Science Foundation. eBlocks are intended to



Figure 1: Approximate size of “Smart Dust” sensor nodes (photo courtesy of Joe Kahn).

enable regular people, having no electronics or programming experience, to construct basic but useful customized sensor-based systems. One such system might alert a homeowner that their garage door has been left open at night. Today, a garage-open-at-night system can be purchased off-the-shelf, but due to low sales volumes, may cost about \$75 – more than many people are willing to pay. Furthermore, the off-the-shelf systems can’t be customized easily, such as being extended to monitor two or three garage doors, or providing alerts at multiple locations throughout the house. In contrast, with eBlocks, the homeowner could purchase a contact switch sensor block, a light sensor block, a logic block, and an LED (light-emitting diode) block. As illustrated in Figure 2, the user could then connect the blocks, and configure the logic to detect the condition of the contact switch sensor and the light sensor each outputting false. The homeowner could easily customize the system for multiple garage doors by adding more contact switch sensor blocks and more logic blocks, could customize the system for multiple alerts by adding a splitter block feeding multiple LEDs distributed throughout the house, or could choose a different alert method by replacing the LED block by perhaps a buzzer block.

As another example, a homeowner might want to set up a system that detects that their child is sleepwalking in the dark. Figure 3(a) illustrates such a system involving a motion sensor block and a light sensor block, feeding into a logic block detecting the motion sensor outputting true and the light sensor outputting false, wirelessly feeding into an LED or buzzer block. Another example would be a daytime doorbell as shown in Figure 3(b), involving a button block, light sensor block, and a logic block detecting the button

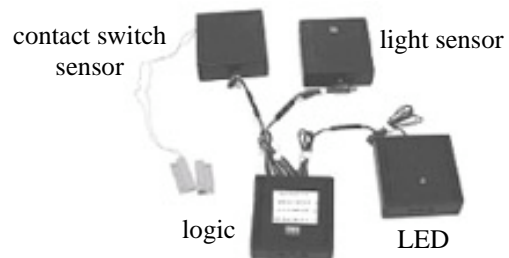
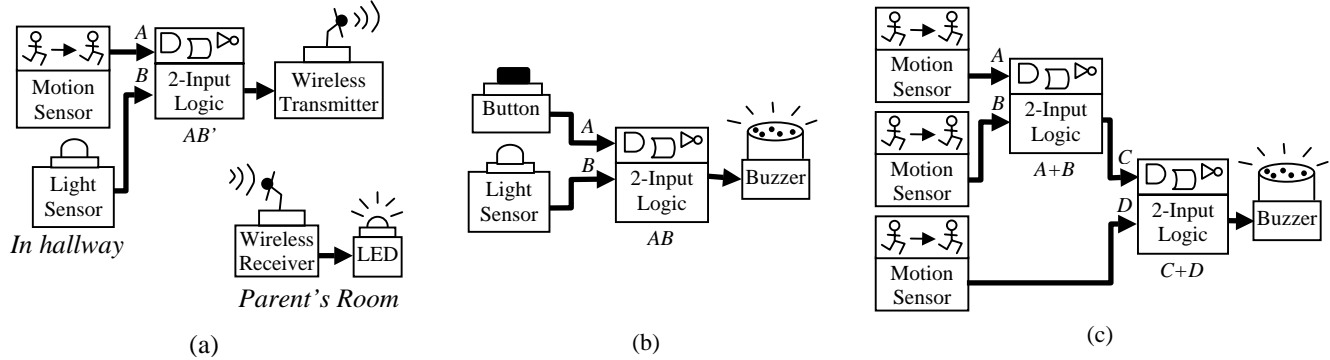


Figure 2: Garage-open-at-night system built using eBlock prototypes.

Figure 3: Various applications built with eBlocks, (a) Sleepwalking Detector, (b) Daytime Doorbell, (c) Motion on Property Alert.



sensor and light sensor outputting true, feeding into a buzzer block. Yet another example would be a system that alerts a homeowner if there is motion anywhere around their home shown in Figure 3(c), involving motion sensor blocks, logic blocks detecting if any motion sensor block is outputting true, feeding into an LED or buzzer blocks. Countless similar uses of sensor blocks around the home exist. Furthermore, such blocks could be useful in offices or stores. For example, a useful office application would indicate whether a conference room was in use. Employees might utilize multiple motion sensors as well as sound sensors inside the room, all of which could be fed into logic blocks computing the OR of all the sensor outputs, feeding into an LED outside the room.

Notice that all these systems can be built from many of same basic building blocks: sensors (of motion, light, sound, etc.), logic, and output (LEDs, buzzers, etc.). Hence, makers of such blocks could sell the blocks in large volumes, bringing the cost of each block down to just a few dollars or even cents each [8].

Notice that all of the above systems require the user to use logic blocks to compute basic Boolean logic functions. For programmers and engineers, AND, OR and NOT form the basic logic blocks from which any logic function can be computed. However, regular people who do not know programming or electronics may have a hard time with these blocks.

In this paper, we will describe several alternative designs for logic blocks, and will provide data from several experiments demonstrating that a sentence-based block yields best results.

TABLE-BASED LOGIC BLOCK AND INITIAL INFORMAL EXPERIMENTS

As we defined different types of eBlocks and developed sample eBlock-based applications, we determined that having a single “logic block” would be beneficial for users. One benefit would be that of fewer blocks in the resulting system, meaning less physical space, fewer connections, and less power consumption. Thus, while we may make traditional AND, OR and NOT blocks available, we also set out to define a general purpose “logic block,” having two

inputs (A, B) and one output, able to carry out most two-variable logic functions that we described in the sample eBlock applications.

We designed a logic block with a 2-input 1-output truth table on the front of the block, as shown in Figure 4. We believed that non-experts would not have trouble understanding a truth table, because a truth table simply lists each input condition, and thus users would merely need to select the desired output (yes or no) for each input condition. For example, for the input condition A=true and B=no, the user would select whether the output should be true or false simply by moving the corresponding DIP switch pin to the true or false position. The user would do this for the other three input combinations too. We liked the truth table design because a truth table can represent all possible Boolean functions of two inputs.

A key design criterion for all eBlocks is that the blocks should be self-explanatory. In particular, no special training including reading a separate set of instructions or taking a tutorial, should be required to use the blocks. Instead, most adults, and even adolescents, should be able to easily understand what each block does merely by reading the name of the block, or possibly by reading a very short (about 10 word) description included on the front of the block. All of our experiments have confirmed to us that non-experts often do not read or understand instructions, even when written on the back of each block. This is consistent with other studies - teenagers involved in computing courses preferred exploratory learning and hated reading even the shortest manuals [18], visitors at museum exhibits ignored any instructions more than 20 words and instead try to work things out for themselves if possible [5].

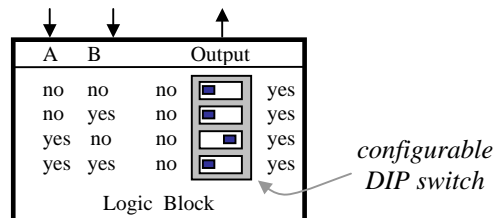


Figure 4: Original eBlock logic block.

Figure 2 shows a picture of physical eBlock prototypes that we built. Each block is about the size of a deck of cards, contains a PIC microcontroller (<http://www.microchip.com>) costing less than one dollar, and operates on 9-volt battery for over two years. We used “yes” and “no” to represent logic values, and included short descriptive phrases on each block. For example, the motion sensor block says “Outputs ‘yes’ when motion is detected, ‘no’ otherwise.”

With these prototypes, we conducted a series of informal experiments in the early part of 2003 wherein we gave users a set of blocks and asked users to build some specific systems, starting with a doorbell (a button connected to a beeper), then a motion buzzer (a motion sensor connected to a buzzer), and then a motion at night buzzer (a motion sensor and light sensor connected to a logic block connected to a buzzer). We observed each user silently for 30 minutes. Participants in these experiments were acquaintances, including neighbors, kids, and friends, and one high-school class. We had planned to conduct formal experiments with larger numbers of high-school and college students later. The participants included a 10-year old female, a 13-year old female, a 40-year old male high-school math teacher, a 75-year old male retired electrical engineer, a 37-year old female homemaker with bachelor’s degree in humanities, a 19-year old female college student studying to be a school teacher, and 20 high-school juniors (male and female) in a chemistry class.

The purpose of our informal experiments was for us to see what aspects of eBlocks people readily understood, and what aspects were challenging. We observed that most of the participants could *not* successfully use the logic blocks without training. Only one of the participants correctly configured the logic block to detect motion at night, and that configuration was through random guessing until the system worked. In our observations, we noted much confusion as to how to configure the DIP switch. We heard many comments like “What do these do?” or “How do these work?” (we did not answer these questions until after the experiment). More than half the participants resorted to random guessing, moving the logic block’s DIP switch pins left or right and then seeing how the system behaved by activating the sensors (waving a hand in front of the motion sensor, or covering/uncovering the light sensor) and listening for sound from the buzzer block. Of those who used random guessing, all but one failed to completely configure the logic correctly. Three other participants tried

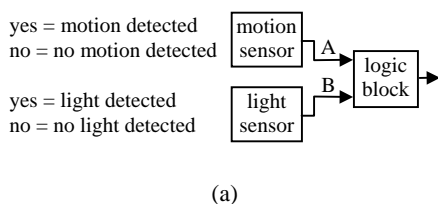
to take a more methodical approach, and verbalized hypotheses as to how the logic block worked (we asked participants to think out loud if possible). One (the retired engineer) initially thought the block was an AND gate, and didn’t understand the purpose of the DIP switches – he tried different configurations of the DIP switches followed by observing how the system worked for each configuration in an effort to understand the block, but gave up after 7 minutes, saying he was “confused about this block.” Another (the math teacher) thought each row applied to *either* of the row’s input conditions, rather than *both* of the row’s input conditions; he thus configured the switches to what he thought was a correct configuration, and then was puzzled as to why the system did not behave properly. Two others (the homemaker and the college student) were reading the table vertically instead of horizontally, leading to no viable hypotheses.

These early informal experiments led to us to focus on creating a better logic block, resulting in a series of formal experiments and logic block versions that we now describe.

Truth Table Experiments Using Written Quizzes

We conducted an experiment in the summer of 2003 to determine whether a truth table based block was intuitive to non-experts. We gave a written quiz during lab sections of a university course on computer applications (e.g., word processing, spreadsheets, etc.), an elective for people in non-science and non-engineering majors at our university. Typical majors of students in the course were psychology, history, business, etc. We introduced the quiz during course lab sections as an anonymous survey, and thus worth no points and optional. 28 of the 29 students agreed to take the quiz. Students had 15 minutes to take the quiz. All quizzes were given by the same graduate student who said very little about the quiz beforehand as to avoid influencing results of one lab section over another. We point out though, the participants demonstrated little interest in the quiz and that many seemed to rush through – meaning that results may be lower than a motivated eBlock user.

We randomly distributed different versions of the quiz among the students, who were not aware of the different versions. All versions first introduced the idea of a motion sensor, a light sensor, and a logic block, using a one-paragraph description and the figure shown in Figure 5(a). All versions then asked the students to detect motion at night. One quiz version allowed the students to answer by



... The logic block contains a switch that enumerates all possible inputs. Check the boxes in which the logic block should output a yes when these inputs are detected.

1. Detect motion at night:
- | | | |
|----|----|--------------------------|
| A | B | <input type="checkbox"/> |
| A | B' | <input type="checkbox"/> |
| A' | B | <input type="checkbox"/> |
| A' | B' | <input type="checkbox"/> |
- (b)

Figure 5: Written quiz to evaluate understanding of truth tables: (a) block diagram of sensors and logic, (b) logic configuration question.

Table 1: Results of written quizzes for truth tables and Boolean equations.

Question	Truth table with variables (11 students)	Truth table with English (9 students)	Boolean equations (9 students)
Motion at night	36%	22%	11%
Motion	0%	56%	0%
Motion at night or no motion in day	0%	22%	0%
Motion or night	0%	11%	0%

checking boxes in a truth table for input combinations that should result in a true output, representing the input combinations as Boolean functions of the input variables, as shown in Figure 5(b). That truth table form matched the original eBlock logic block design. Table 1 summarizes results in the *Truth table with variables* column. 11 students received that quiz version. Only 36% of the students correctly filled in the table outputs for the motion at night problem. Furthermore, the quizzes included additional questions asking students to detect other sensor conditions, including just detecting motion, detection motion at night or no motion in the day, or detecting motion or nighttime. None of those 11 students correctly configured the table’s outputs for those more difficult conditions.

One might assume that the reason for the low scores was due to the use of variable names (A, B) and the complement symbol (A’, B’), rather than inherent difficulties understanding a truth table. To test this assumption, we used a second version of the quiz that also used a truth table, but instead of representing the input combinations using Boolean functions of the input variables, that version wrote out each input combination in English using the sensor functions, as shown in Figure 6. Such a block would not be a feasible eBlock, because the company designing a block does not know a priori what sensors will be connected to the block’s inputs – the logic block might be used with other sensors, like a water sensor and a contact switch sensor. Nevertheless, we included this block version to test whether the problem lied in the variables names or the truth table itself. As can be seen in Table 1 in the *Truth table with English* column, students still performed poorly. Only 22% correctly configured outputs for the motion at night question. The table shows that success was slightly better than the variable-based table for the more complex questions, but still quite low.

We also checked whether people were comfortable writing Boolean equations, which might imply the need for a text entry based logic block. We used a third quiz version that showed an example of a Boolean equation of A and B, and that asked students to write the correct Boolean equation of

Figure 6: Quiz answer form for truth table using English.

1. Detect motion at night:

motion	light	<input type="checkbox"/>
motion	no light	<input type="checkbox"/>
no motion	light	<input type="checkbox"/>
no motion	no light	<input type="checkbox"/>

input variables. As seen in the Table 1 in the *Boolean equations* column, success rates were low.

Just to be sure that the questions posed in the quizzes were understandable and that the notion of truth tables was the item causing difficulties rather than the structure of the questions, we gave the quiz version using a truth table with variables to students in a lab section of a second course in digital design. Such students had therefore seen truth tables in the first course in digital design. All six students agreed to take the no-credit quiz, and the six students in that lab (a small number due to the course being a summer offering) did very well – 100% answered the motion at night question correctly, and the average success on the remaining three questions was 90%.

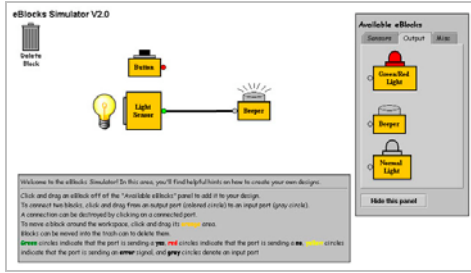
In talking with the non-expert students who took the quizzes, we found many simply had no understanding of what the rows and columns of the table signified – for example, some didn’t know whether to read left to right or top to bottom, and some of those who did correctly read left to right didn’t assume that the row’s output only applied when the input conditions of that row were both true. We concluded that a truth table is not a known concept for non-experts. People did quite poorly using the truth table with variables. Even if we replaced the variables with English descriptions of the variables’ meanings, people still performed rather poorly. We thus began our search for a single-block logic design that was more readily understandable by users.

TRUTH TABLE AND SENTENCE EXPERIMENTS USING A SIMULATOR

Simulator

To support our next series of experiments, we developed an eBlock simulator. The simulator enabled us to experiment with larger numbers of users than possible with our limited number of physical prototypes. At the same time, the simulator also enabled the feature of users being able to examine the behavior of their configured systems – a feature that exists when eBlocks are used in practice, but that is absent in written quizzes. The simulator, as shown in Figure 7, had a panel on the right containing icons of each available eBlock type. A user could drag an icon from the panel to a canvas on the left, instantiating a new copy of the block. A user could drag a wire from a block’s output to another block’s input to create a connection. Each type of sensor block had a method for the user to create the sensed event – a user could press the button on the button block, turn on the light in front of a light sensor block, etc. Each type of output block generated visible and/or audible output – an LED block would light up red or green, and a beeper

Figure 7: Web-based eBlock Simulator.



output block would create an audible beep as well as a graphical animation of sound (lines projecting radially outward from the block).

New Logic Block Versions

Convinced that a traditional truth table was not viable for our logic block, we proceeded to define three “improved” versions of table-based logic blocks, and a sentence-based logic block. These four versions, as they appeared in the simulator, are shown in Figure 8. We named the blocks “Combine” rather than “Logic Block” as we found in our earlier discussions with participants that most did not understand the meaning of the word “logic” in the context of eBlocks. We now briefly describe each logic block version.

Phrased truth table

Figure 8(a) shows the most basic truth table we used for these experiments. We replaced “yes” and “no” input column entries by phrases like “A is yes” and “B is no.” We also added a sentence to the right of the block saying “The output should be yes when:.” We expected this block perhaps to perform only slightly better than a traditional truth table that has just “yes” and “no” column entries.

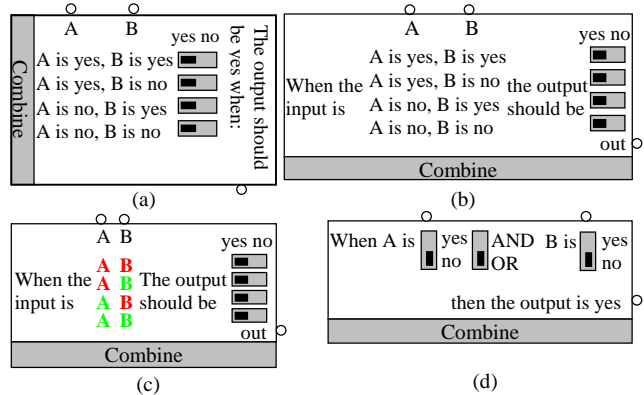
Phrased truth table embedded in a sentence

Figure 8(b) illustrates a version we hoped would perform much better than the phrased truth table. We reformatted the block to look more like a sentence, and we embedded the truth table within that sentence-like format. The sentence began with “When the input is,” followed by the phrased truth table that could be interpreted such that each row continued the sentence (e.g., “A is yes, B is yes”), followed by “the output should be,” followed by the DIP switch with “yes” and “no” at the top to complete the sentence. We intentionally used a comma between the truth table’s input phrases, rather than using the word “and.” Using this comma resulted in the sentence not being a correct English sentence, but we felt that including the word “and” would confuse people who did happen to understand the concept of a logical AND, who might thus think the logic block represented an AND function.

Colored truth table embedded in a sentence

In our various experiments with physical prototypes, we observed users paying close attention to small green and red

Figure 8: Logic block versions used in our first simulator experiments: (a) phrased truth table, (b) phrased truth table embedded in a sentence, (c) colored truth table embedded in a sentence, (d) logic sentence.



LEDs that we had placed at every block’s output. Each block would blink either its green or red LED when transmitting a yes or no packet. We had originally placed these LEDs for our own debugging purposes, but after observing users using these LEDs for their own debugging purposes, we decided to include the LEDs as a permanent feature on the blocks, including on the blocks in our simulator. We then hypothesized that users might develop a “feel” for green meaning yes and red meaning no, and thus we thought of the idea of replacing the truth table’s input condition phrases, like “A is yes” or “A is no,” simply by the letter “A” colored green or red.

Figure 8(c) illustrates a version of the truth table embedded in a sentence, using a colored truth table. The first row’s letters are both red, the second row has A red and B green, the third row has A green and B red, and the fourth row has both letters green.

Logic sentence

Figure 8 (d) shows a logic block format in which we go fully with a sentence structure and we eliminate the use of a truth table entirely. Instead, we use a configurable logic sentence, where the user can move two DIP switches that change the polarity of each input variable in the sentence, and can move another DIP switch to change whether the inputs are ANDed or ORed. We embed these DIP switches within the sentence so that user can easily read the sentence that they configured. We also take care to make sure that the input connections appear right above the input name in the sentence, eliminating the name to have “A” and “B” to appear twice on the block. We expected this block to perform much better than any of the truth table versions. However, the drawback of this logic block version is that this version does not support all possible Boolean functions of two inputs, instead supporting only eight of the possible 16 functions. Notably missing are XOR and XNOR (exclusive OR and exclusive NOR, respectively).

Table 2: Results of simulator-based experiments for improved truth tables and logic sentence.

Question	Phrased truth table (33 students)	Phrased truth table embedded in sentence (30 students)	Colored truth table embedded in sentence (32 students)	Logic sentence (32 students)
Daytime doorbell (AB)	15%	20%	22%	38%
Garage open at night (A'B')	16%	13%	25%	19%

Experiments

Simulator-based experiments – four logic block versions

We created a modified version of the simulator that could contain a pre-designed eBlock system with the logic block in the center, so that the user only had to configure the logic block to complete the system – the user did not have to instantiate or connect blocks. Furthermore, we created a unique version of the simulator for each logic block version – each simulator version contained only one of the logic block versions. We assigned simulator versions to the students randomly.

We created pre-designed systems for two different problems, with instructions briefly describing the problem and asking the user to configure the switches on the “combine” block to correctly solve the problem. Students could proceed to the next problem by pressing a “next” button. The first problem was a daytime doorbell, requiring a button sensor, light sensor, and a logic block configured to compute the function AB. The second problem was a motion at night system, requiring a motion sensor, light sensor, and a logic block configured to compute AB'.

We conducted experiments in the spring of 2004 using the simulator. The participants were 127 students in the lab sections of our earlier-mentioned computer applications course, and also our introduction to programming course for non-scientists/non-engineers, whose students' majors are almost identical in makeup to the computer applications course. Students in non-science/non-engineering majors must take the computer applications course, the programming course, or a math course, so we might conclude that participants were not particularly interested in computers. As this was a different quarter than when we gave our earlier written quizzes, the students had not seen eBlocks before. We again introduced the quiz minimally, indicating that the quiz was anonymous, not for credit and voluntary. All 127 students took the quiz. Again, due likely to the lack of our saying much about the experiment, most participants seemed disinterested. Students had 15 minutes

Table 3: Results of simulator-based experiments for logic sentence and colored truth tables. Numbers in parentheses include students whose answers were “close to correct.”

Question	Colored truth table embedded in a sentence (15 students)	Logic sentence (17 students)
Daytime doorbell (AB)	47% (67%)	47% (71%)
Nighttime doorbell (AB')	33% (52%)	41% (76%)
Motion on property (A+B)	33% (33%)	65% (71%)

to complete the simulator-based quiz; most finished in less than 10 minutes.

Results – four logic block versions

Table 2 summarizes results. We see that the logic block versions having truth tables embedded in a sentence slightly outperformed the phrased truth table. We also see that the logic sentence version seemed to outperform the truth table versions, demonstrating 38% success for the daytime doorbell.

Simulator-based experiments – table versus logic sentence

We ran simulator-based experiments again in the summer of 2004, with two-thirds of the 32 participants coming from the previously mentioned computer applications and programming courses, and one-third coming from our introduction to programming course for scientists and engineers. Again, all students were new to eBlocks. We again said little about the experiment and we observed students to be somewhat disinterested. We included three problems, a daytime doorbell whose correct logic computes AB, a nighttime doorbell whose correct logic computes AB', and a motion on property system having two motion sensors and whose correct logic computes A+B (A or B). We tested two logic block versions: the colored truth table embedded in a sentence, and the logic sentence. The experiments this time did not pre-instantiate the required blocks; participants had to instantiate and connect the sensor, logic and output blocks themselves. Students had 15 minutes to complete the exercises.

Table 3 summarizes results. We see that about half of the students (47%) correctly configured the logic for the daytime doorbell using either logic block version. We see slightly better performance by the students using the logic sentence block for the slightly more difficult nighttime doorbell. Most interesting perhaps is that we see significantly better performance by students using the logic sentence block for the motion on property problem – 65% success, versus 33% for the truth table block. Although that problem seems as simple, the problem requires a student using a truth table block to place three switches in the yes position, whereas the other two problems require only one switch in the yes position and thus are likely simpler conceptually to understand. In contrast, the sentence

Table 4: Results of simulator-based experiments for logic sentence and colored truth tables with slightly more-advanced students. Numbers in parentheses include “close to correct” solutions.

Question	Colored truth table embedded in a sentence (12 students)	Logic sentence (13 students)
Daytime doorbell (AB)	42% (67%)	92% (100%)

structure only requires moving the function switch from AND to OR.

We also examined the students’ answers for answers that were “close to correct.” For the truth table block, we defined close to correct as meaning that only one switch was in the wrong position, or for a solution that required only one switch to be in the yes position, then the wrong switch was in the yes position. For the sentence block, we defined close to correct as meaning that one of the input switches was in the wrong position; if both input switches were in the wrong position, or if the function switch (AND/OR) was in the wrong position, we did not consider the answer close to correct.

Table 3 shows the percentage of students getting answers correct or close to correct, with those percentages shown in parentheses. Notice that the logic sentence outperforms the truth table for all three problems. Particularly noteworthy is that for the motion on property problem, none of the students who failed to configure the table correctly were even close to correct.

We conducted the same experiment in a second course in programming, where the students had all completed a first course in programming for scientists/engineers. Thus, we don’t consider these students to be “non-experts” as they are likely majoring or minoring in a computing or other engineering major. Also, we gave students only one logic configuration problem, namely the daytime doorbell (the reason being that we also gave them problems related to building systems with state using state-based eBlocks, discussion of which is beyond the scope of this paper). Table 4 summarizes results. We see that while only 42% of the students successfully configured the truth table block, 92% (all but one student) configured the logic sentence correctly.

These results are very intriguing. They tell us that even having an interest in majoring in computing (68% of the participants were some form of computing major – all other majors were other engineering or science majors) and having programming experience as demonstrated by completing a first course in programming, do not improve the success rate of using the truth table logic block. In contrast, such interest and experience greatly improves the success rate of the logic sentence block to nearly 100%, which makes sense because the logic sentence block looks

Table 5: Results of simulator-based experiments for logic sentence block, with motivated high-school graduates planning to study engineering. Numbers in parentheses include “close to correct” solutions.

Question	Logic sentence (8 students)
Daytime doorbell (AB)	88% (88%)

just like a Boolean expression in a programming language’s branch or loop construct.

Simulator-Based Experiment -- Logic Sentence Block and Motivated Participants

We conducted an experiment in the summer of 2004 in which the participants were eight high-school graduates planning to attend our university as some type of engineering major, who voluntarily enrolled in a summer enrichment program that met every Friday. On our assigned Friday, we first gave a one-hour talk to these students about the field of engineering in general and about college. After a lunch break, we met with the students in a computer lab, where we were to have a two-hour hands-on session. Thus, unlike our previous experiments, we characterize these participants as having great interest, partly because their participation was entirely voluntary, because they knew that learning about eBlocks was the purpose of their next two hours, and because we had the opportunity to build a basic relationship with the students during the first hour.

We gave them the simulator-based experiment, beginning with the daytime doorbell as the problem to solve (other problems involved state-based blocks and are beyond the scope of this paper). Students had to instantiate and connect all blocks and had to correctly configure the logic block. The only logic block available to them was the logic sentence block.

Table 5 summarizes the result. Seven of the eight students, or 88%, correctly instantiated and configured the logic sentence based logic block.

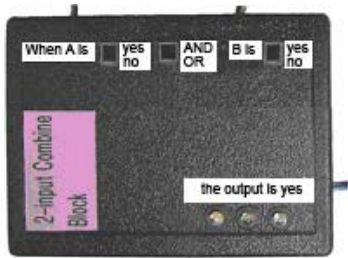
NEW PHYSICAL LOGIC BLOCK PROTOTYPE BASED ON THE LOGIC SENTENCE

Based on the results of experiments described above, we built new physical prototypes for our logic block, using a logic sentence structure. Figure 9 shows a photograph of one of those blocks.

We have informally utilized these blocks with 10 additional users, and have not yet seen users encounter any problems utilizing the block.

Although a logic-sentence block only covers eight of the possible 16 functions of two inputs, we have found that those eight functions seem to cover most practical uses of a logic block. As evidence supporting this statement, we examined a set of eBlock applications we had developed in early 2003, before embarking on our search for a new logic block design. Each application consisted of anywhere from

Figure 9: Physical prototype based on a logic sentence.



two to a dozen blocks, averaging about six blocks. We developed the set to represent typical uses around the home, office or store. At the time, we assumed we would be using a truth table based logic block, able to compute all functions of two inputs. After developing the new logic sentence block just recently, we went back to see if the block could replace the truth table based block we had previously used in the applications. Seven of the twenty-three applications required logic blocks. All seven could have their single truth table logic block replaced by a single logic sentence based logic block. Furthermore, we could not think of any common eBlock applications utilizing XOR or XNOR functions.

OTHER EXPERIMENTS

Comparison with AND/OR/NOT blocks

Earlier in the paper, we described our rationale for wanting a single configurable logic block, rather than (or in addition to) separate blocks for AND, OR and NOT. In short, we want to minimize the number of blocks in final eBlock systems, since each block takes space and consumes power. Furthermore, a single logic block has the advantage of enabling a user to change the configuration without changing blocks – for example, a user could change a doorbell into a daytime doorbell temporarily just by moving some switches, rather than adding or removing blocks.

Nevertheless, we felt it was important to compare the user success rate of our single block logic approach with an approach using separate blocks for AND, OR and NOT. Thus, in our earlier described simulator-based experiments summarized in Table 3 and Table 4, we had also included AND/OR/NOT blocks as one of the versions. Those experiments required users to instantiate, connect, and configure the blocks. The AND/OR/NOT version of the simulator had an AND block, an OR block, and a NOT block. Users of that version just had to instantiate and connect the blocks, as those blocks don't require configuration.

Table 6 summarizes results. We see that the AND/OR/NOT blocks are competitive with the best truth table and logic sentences. However, we see that the logic sentence still seems to have a higher success rate – comparing to Table 4, we see that the AND/OR/NOT version had a success rate of 69%, compared to the logic sentence of 92%. Nevertheless,

Table 6: Results of using AND/OR/INV blocks. Numbers in parentheses include “close to correct” solutions.

Question	AND/OR/INV blocks (16 students) – compare to Table 3	AND/OR/INV blocks (13 students) – compare to Table 4
Daytime doorbell (AB)	63% (69%)	69% (77%)
Nighttime doorbell (AB')	50% (38%)	
Motion on property (A+B)	63% (63%)	

AND/OR/NOT seems to be a viable option when multiple blocks for logic are feasible.

Colored logic sentence

We had also experimented with a colored logic sentence logic block. Inspired by the success of the colored truth table over the phrased truth table, we thought perhaps a colored logic sentence might exhibit similar improvement over a logic sentence. Figure 10 shows the block version we tried. The top letter A and letter B were colored green, and the bottom letter A and letter B were colored red. Notice that the block has fewer words than the block in Figure 8(d), which we thought might lead to easier comprehension.

We had introduced this colored logic sentence block into the earlier-described experiment whose results appear in Table 4. 12 students had received the simulator version with the colored logic sentence block. The colored logic sentence block did not perform as well as the regular logic sentence block, having only a 58% success rate compared to the 92% success rate of the regular logic sentence block.

RELATED WORK

Sensor-based Systems

There is much work that strives to simplify the design and integration of sensor based components. The intended audience ranges from pre-school aged children to engineers, from educational aids to engineering solutions. We briefly discuss three classes of solutions: programmable, board-based, and block-based.

Programmable

Programmable products are composed of a programmable board or block to which a user can easily add desired sensors and actuators. Programmable products are applicable to many different situations and must ensure that the user can successfully specify a program's functionality.

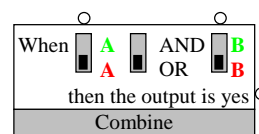


Figure 10: Colored logic sentence logic block.

MIT Crickets [14] and GoGo Boards [13] are intended for science education. A user programs a centralized board to perform a variety of functions using a variation of the Logo language called Cricket Logo. Two types of programming environments are provided. LogoBlocks is a graphical environment in which users drag icons from a pallet and stack the icons together to form a program. In terms of logic based blocks, users can choose from AND, OR, and NOT blocks. Jackal is a text based programming environment; for logic, users can use keywords AND, OR, XOR, and NOT.

Lego Mindstorms [10] are toys intended for robotics applications. Users program the processor using a simple graphical language in which users drag icons from a pallet. No explicit logic operators are available. Instead users must cascade if statement icons to form various AND and OR constructs. Negation can be implemented by graphically selecting the state of a sensor (e.g. light sensor state choices are bright, dark, blink, and range) within the if construct.

Phidgets is a programmable product aimed at industrial applications [17]. Users program Phidgets using Visual Basic. Similar to many programming languages, operators include AND, OR, NOT, and XOR. Teleo [20] is a similar product that can be programmed utilizing a variety of languages and utilizing similar logical operators.

Mica Motes [7] and Smart Dust [21] nodes contain a multithreaded operating system called TinyOS. Applications are written in NesC, which has a C-like syntax. Applications can then be compiled and downloaded to various boards. A collection of nodes are capable of self-configuring a multi-hop network and support dynamic reprogramming within the network.

Requiring a user to learn a programming language may intimidate non-expert users and conflict with one of the main goals of eBlocks, thus alternative solutions must be examined. For users comfortable with programming, we intend to make available a programmable block that can be used in conjunction with other blocks, and that is not a required component to construct a given eBlock application.

Board-Based

Board products consist of electronic components that must be connected on top of a specialized circuit board typically intended to provide power to the individual components.

MagicBlocks [9] were designed to teach pre-university students basic logic theory, before the students begin university level computer science classes, by building a variety of systems. The MagicBlocks kit contains a logic block, which can be configured via a button to compute either a logical AND or OR.

Block-Based

Block products are composed of electronic components that can be connected together to build the desired system and

do not require a central module or specialized circuit board to implement the systems. Users simply need to connect the desired blocks together to build complete systems.

Logidules [12] were designed to help university level students studying electronics to build hardware systems. Using Logidules, students snap together boxes that represent a range of components from logic gates to microprocessors. A Logidules simulator is also available [4]. Specific logic blocks include AND, NAND, NOR, XOR, and INVERSE.

Logiblocs [11] is a product consisting of small plastic building blocks geared for education and educational toys. Three types of logic blocks are utilized - AND, OR, NOT. The AND and OR blocks are 2-input, 1-output.

Electronic Blocks [22] are blocks that consist of processors incorporated inside of LEGO Duplo Prima™ blocks intended for children aged 3 to 8. Users simply stack the correct combination of Lego blocks to produce the desired output. Two types of logic blocks are included, invert and AND.

As mentioned previously, while the use of separate AND, OR, and NOT is feasible, we want to minimize the number of blocks required to build various eBlocks systems to reduce power and cost.

Expressing Boolean Relationships

The difficulty of expressing Boolean equations is not limited to sensor networks. A survey of front-end approaches for searching online public access catalogs found composing Boolean expression for searches to be difficult [6]. Users were unable to compose Boolean expressions and required the aid of an expert. Web search engines also posed problems for users. With over a billion webpages, users must rely on search engines to find the desired information. However, 7 out of 10 users are dissatisfied with Internet search engines and less than 6% of users use Boolean search terms “and”, “or”, “+”, and “-” [19]. Users are familiar with AND and OR because they are found in natural language (i.e. English) but these words take on a different meaning when used in a Boolean search. The errors occurs because users rely on their knowledge of English and substitute the AND logical operator for the OR logical operator. Users are also unfamiliar with the scope of the NOT operator varies and often ignore parenthesis [16].

An information retrieval system, which aims to aid in the construction of Boolean equations, is presented in [2]. A natural language query specified by the user is translated into tiles, vertical tiles symbolize terms that are ORed and the horizontal tiles are ANDed. The user is then able to view how the query is interpreted and able to directly manipulate the tiles by activating/deactivating tiles, dragging tiles from one column to another, or expanding and contracting across columns, and updating the search terms themselves. This allows the user to control how the query is interpreted.

A similar format to formulate Boolean queries is proposed in [16]. Match forms or cards are provided in which a user specifies search terms within the slots provided and can specify a negation by prefacing the term with NOT. All terms within the form results in a conjunction (AND). disjunction (OR) is specified by adding match forms.

A graphical filter/flow representation of Boolean equations using a water metaphor was presented in [23] to aid users in specifying Boolean equations specifically for querying databases. Users formulate and express queries by representing the logical representation of the query in the form of water flowing through filters. This representation is hypothesized to work better than traditional linear text specification because users can relate prior knowledge of the domain (specifically water flowing through filters) to the Boolean query.

While many alternatives for specifying Boolean equations can be found, they do not translate well to the logic block interface. eBlocks are limited by power, cost, and physical dimensions, so a large graphical interface is not feasible. Further, we do not want to require a computer to configure the various blocks.

CONCLUSION

We have described an emerging class of sensor networks, which requires non-expert users to specify Boolean logic equations. We have presented a variety of logic block interfaces and through a series of experiments we have shown that non-expert users have much difficulty with a block based on Boolean logic truth tables. We demonstrated that utilizing color in truth tables improves success. We also showed that a logic block having a sentence-like structure with some configurable switches yields a better success rate.

REFERENCES

1. Akyildiz, I. F., Su, W., Sankarasubramanian, Y., and Cayirci, E. Wireless sensor networks: a survey. *Computer Networks* 38 (2002), 393-422.
2. Anick, P. G., Brennan, J. D., Flynn, R. A., Hanssen, D. R., Alvey, B., and Robbins, J. M. A Direct Manipulation Interface for Boolean Information Retrieval via Natural Language Query. *Proc. ACM SIGIR Conf. On Research and Development in Information Retrieval, User Interfaces* (1990), 135-150.
3. eBlock: Embedded Systems Building Blocks. <http://www.cs.ucr.edu/~eblock>, 2004.
4. Ecole Polytechnique Federale De Lausanne. Logidules Online. <http://lspcc51.epfl.ch/logidules/>
5. Gammon, B. Everything we currently know about making visitor-friendly mechanical interactives. British Interactive Group, <http://www.big.uk.com>, 1999.
6. Hidreth, C. R. Intelligent Interfaces and Retrieval methods for Subject Search in Bibliographic Retrieval Systems. *Advances in Library Information Technology*, 2 (1989).
7. Hill, J., D. Culler. MICA: A Wireless Platform For Deeply Embedded Networks. *IEEE Micro* 22, 6 (2002).
8. Intel Corporation. Instrumenting the World. http://www.intel.com/research/exploratory/instrument_world.htm, 2004.
9. Kharna, N. and L. Caro. MagicBlocks: A Game Kit for Exploring Digital Logic. *Proc. of the 2002 American Society for Engineering Education Annual Conference* (2002).
10. Lego Mindstorms. <http://mindstorms.lego.com>
11. Logiblocs. <http://www.logiblocs.com>.
12. Logidules, <http://diwww.epfl.ch/lami/teach/logidules.html>.
13. MIT Media Laboratory. GoGo Board. <http://learning.media.mit.edu/projects/gogo/index.html>
14. MIT Media Laboratory. Programmable Bricks. <http://ilk.media.mit.edu/projects/cricket/>
15. National Research Council. Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers. National Academies Press (2001).
16. Pane, J. and Myers, B. Tabular and Textual Methods for Selecting Objects form a Group. *Proc. Visual Languages* (2000), 157-164.
17. Phidgets, <http://www.phidgets.com/>
18. Sikorski, M. Teaching Computers the Young and the Adults: Observations on Learning Style Differences. *CHI* (1998), pp 42-43.
19. Tanaka, J. The Perfect Search. *Newsweek* 134, 13 (1999), pp 71-72.
20. Teleo, <http://www.makingthings.com/>.
21. Warneke, B., M. Last, B. Liebowitz, and K. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. (2001), pg. 44-51.
22. Wyeth, P. and Purchase, H. Using Developmental Theories to Inform the Design of Technology for Children. *Small Users - Big Ideas: Proceedings of Interaction Design and Children* (2003).
23. Young, D. and Shneiderman, B. A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. *Journal of American Society for Information Science* 44 (1993), 327-339.