I/O Blocks: A Modular I/O Network Framework for Interactive Design Projects

Marutpong C., Phanupong J., Tanaphon, A., Thotsaphon, B., Arnan, S.
Department of Computer Engineering, Faculty of Engineering,
Chiang Mai University
239 Huaykaew Rd
Muang, Chiang Mai, Thailand, 50200

ABSTRACT

I/O Blocks is a platform for tool developers. I/O Blocks consists of nodes, each with a programmable ID and a set of input/output ports. These nodes can be connected to form a grid network. I/O Blocks continuously detects the structure and members of this 2D network. A simple Cartesian coordinate system (X, Y) is used to address and control each node. I/O Blocks allows one to rapidly design projects that requires a modular input/output system where the number of nodes and ports can be added or removed dynamically in real-time. A Python API is available for applications requiring an interface to a computer. Finally, three use case scenarios are presented.

Keywords

Interaction design, robotics, Sensing and Control, Modular, Grid Network Topology.

1. INTRODUCTION

This work has developed a modular I/O platform that simplifies two common obstacles found in interactive project design. First, the number of input and output required vary significantly from project to project. Although microcontroller platforms such as the Arduino, Basic Stamp, and other toolkits in the "breakout-board" category can offer a large number of I/O channels, these numbers are fixed. Therefore, a designer often ends up with either too few or too many ports. In the case where a large number of I/O is required, a multi-board approach is common. Programming this system often involves low-level communication protocols such as I2C, Serial, SPI, etc., which adds an overhead to the development process. With I/O Blocks, instead of letting the designer come up with all the hardware and software workarounds, it offers a modular I/O system together with a software abstraction framework that removes the underlying technical complexity so the designer can focus more on their project objectives.

Secondly, I/O Blocks is useful for projects that require dynamic arrangement of connected "nodes". The spelling game described in 3.2 is a good example. Each I/O block acts as a node with a programmable ID. In a spelling game, each block is assigned an alphabet and the player can put together a word by connecting these individual blocks. The structure of this network of nodes is automatically determined and reported to a host computer, which checks the spelling and reports the result. Again, implementing this ability using off-the-shelf tools would be a time consuming detour for a designer who just wants to get his/her project done.

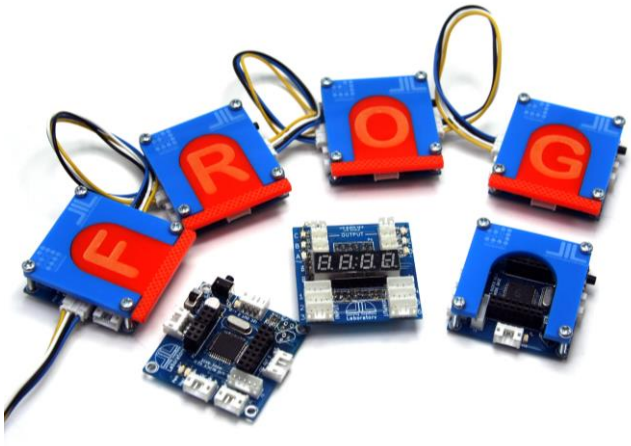

Figure 1: The I/O Blocks System Consists of nodes that can be dynamically connected.

2. The I/O Blocks Architecture

The following sections describe the I/O Blocks system architecture in detail.

2.1 Key Features

The following are the key abilities that we expect the I/O Blocks framework to offer

- Modular design with a maximum of 112 node supported
- Each node consists of multiple analog inputs and digital output ports
- Each node supports the creation of shields. This allows easy customization of what each node can do.
- Supports a gird topology allowing for easy identification of the node location using an (X, Y) Cartesian coordinate system.
- Hot-Plug. Nodes can be added and removed dynamically.
- Each node can be given an ID or name. This string can be used to identify or address the node.
- A master or root node have full access to each node's input and output and the master can be controlled either by a computer or another microcontroller device, such as a GoGo Board.

The following sections elaborate the above capabilities in more detail.

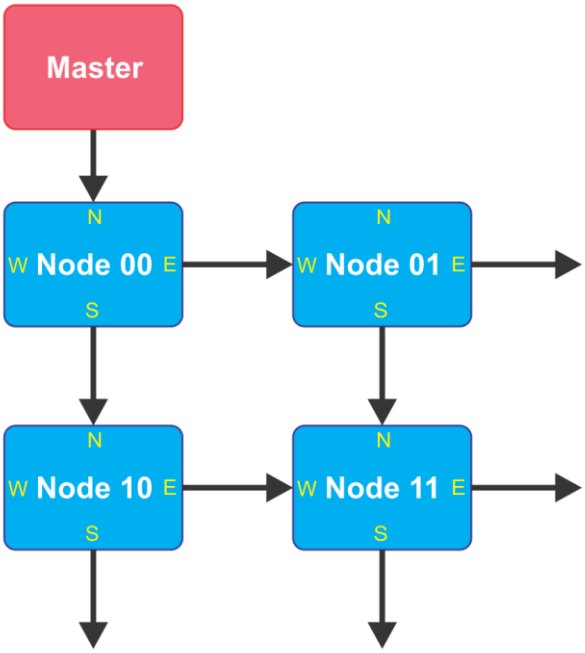

Figure 1. The I/O Blocks grid topology.

2.2 Nodes and the Half-grid Network Topology

I/O Blocks consists of "I/O Nodes" that can be connected to construct a 2D gird network. Each I/O Node consists of four network ports or "gates" as shown in Figure 1. The North and West gates are incoming network ports used to connect the current node to an existing network. The East and South gates are outgoing network ports used to attach more nodes. This arrangement allows the grid to grow in the East and South directions. Therefore, this network topology is considered a "half-grid" as opposed to a "full-grid". The automatic node discovery mechanism in a half-grid network is much simpler than in a full-grid system. The design tradeoffs have been considered and we believe the half-grid approach is sufficient for most of the design needs.

Each node will consist of analog input channels and digital output drivers. These general purpose I/O can be used to create "shields" which stacks on top of the base unit. An example shield has been created and consists of a four-digit 7-segment display and six input and six output ports. Other shields can be designed to serve different needs of a particular design project.

2.3 Dynamic Presence and Location Discovery

I2C is the underlying network system used in this work. It was chosen based on its simplicity, the ability to address multiple nodes,

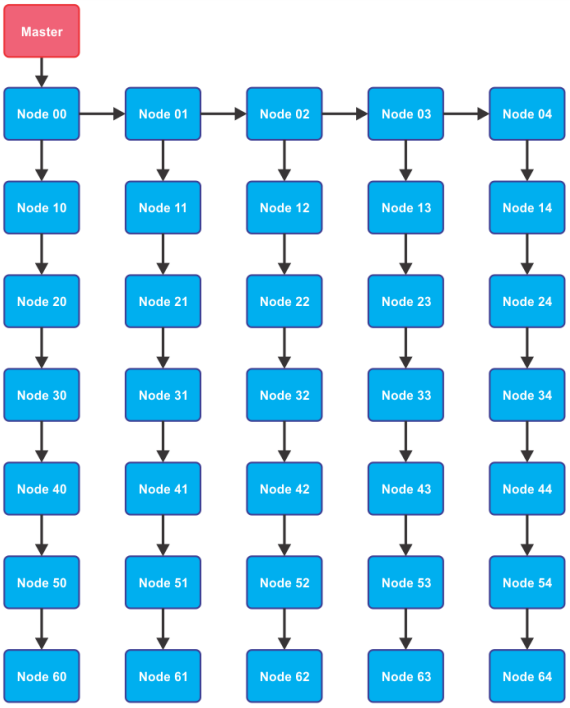

Figure 2. A 7x5 Array Example.

and the existing hardware support in most low-cost micro-controllers. Using I2C allows us to utilize many low-level built-in functionalities that are bundled with the microcontroller such as address recognition, bus collision, and buffering. However, I2C is a bus system that, by definition, cannot identify where each node is located within the network. Therefore, a new mechanism has been added to fix this limitation. Each node consists of MOSFET switches that are placed in series with the East and South side of the I2C data lines. This allows each node to connect or disconnect its child nodes to and from the I2C network. Each node first waits for itself to be added to the network before connecting its own child nodes. This process takes place recursively until the entire network is discovered. This discovery process is coordinated by the "root" node, also called a master block. The master block holds a table that describes the current network structure.

2.4 Node Identification Schemes

There are multiple layers of identification in each node. At the lowest level, each node requires an I2C address in order to send and receive data on the I2C network. This address is given dynamically by the master block to each new node once it is connected. The use of I2C limits the maximum number of logical nodes in the network to 112 (128 available addresses minus 16 reserved addresses [1]).

The second identification layer is a logical 2D grid address. Each node will hold a Cartesian X and Y ID. We have adopted this addressing scheme based on its popularity in identify objects in a 2D space.

The third identification layer is a node-name. This ID serves as a readable property of the node. Once a node has been address by location, the node name or ID string can also be read. While the I2C address can be considered a name, being able to assign and use a string would be friendlier to users. This ID string can be changed by software.

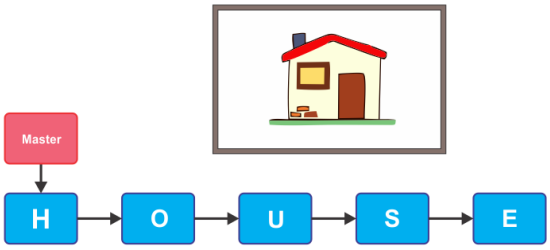

Figure 3. A Spelling Game Example.

3. Usage Scenarios

The following are examples of projects we have created using the I/O Blocks framework.

3.1 Projects with a large number of input and output requirements

We have worked with an interactive exhibition artist who wants to create a light display that reacts to the environment. The artist wanted a 7 by 5 light array, or 35 independently controllable light bulbs. In this scenario, the artist can choose to connect 35 I/O Blocks as a grid each controlling one light bulb. Each bulb can be addressed using its X and Y position and controlled accordingly. Alternatively, since each I/O Block consists of multiple outputs, the artist can reduce the number of I/O blocks used. In this case, seven nodes were used, each controlling 5 bulbs. Since I/O Blocks also have inputs, the system can be connected to 35 sensors which independently affect its corresponding light bulb. This project is currently on permanent display at Chiang Mai University in Thailand. Expanding this project to a higher density of light bulbs should be straightforward both in terms of hardware and software requirements, as long as the number of I/O Blocks nodes is within the 112 maximum range.

3.2 Projects needing to determine node location and allow nodes to be added and removed at runtime

Word games are good examples of this group. For example, if a teacher wants to create a spelling game where a student must snap letter blocks together to spell the word that matches a picture, the teacher can use 26 or more I/O Blocks, each assigned a name using the letters in the English alphabet ("A", "B", "C", Etc.). These name are then etched onto pieces of acrylic and laser-cut to sit on top of each node. When the student puts together a word (by connecting the nodes), the computer checks the number of blocks and their order to figure out if the student has made the correct spelling.

3.3 A Platform for Tangible Programming Languages

I/O Blocks can serve as a platform for those interested in creating tangible programming blocks. The MIT Scratch [2] and Google's Blockly API [3] are two good examples of such programming environment created on a computer screen. I/O Blocks can be used to create a physical version of this programming paradigm such as demonstrated in Robo-Blocks [4]. As in Robo-Blocks, we have created a programming system to control a physical floor robot. The

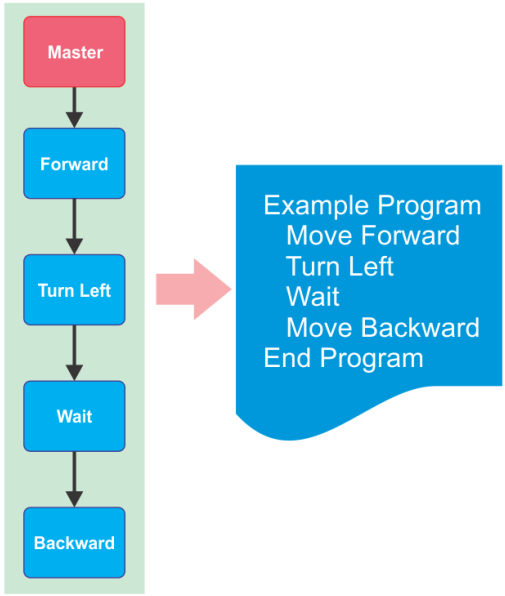

Figure 4. A Tangible Programming Example.

learning activity resembles that in the Logo programming language where learners write programs to control an on-screen turtle to draw various geometrical shapes [5]. Each I/O blocks is assigned a command name (forward, backward, left turn, right turn, and wait). When the blocks are connected to each other, the command sequence can be determined and translated into a computer program.

4. Conclusions

I/O Blocks aims to provide a high-level API for designers to create applications that require a modular I/O system that can determine the network structure and dynamically manage the addition or removal of nodes. The hope is that many types of applications that require these abilities will be simpler to implement and attractive to designers who are not highly technical.

5. References

[1] Semiconductors, P. (2000). The I2C-bus specification. Philips Semiconductors, 9397(750), 00954.

[2] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. Communications of the ACM, 52(11), 60-67.

[3] Fraser, N., Google Blockly – a visual programming editor. URL https://code.google.com/p/blockly/. Accessed Oct, 2014.

[4] Sipitakiat, A., & Nusen, N. (2012, June). Robo-Blocks: designing debugging abilities in a tangible programming system for early primary school children. In Proceedings of the 11th International Conference on Interaction Design and Children (pp. 98-105). ACM.

[5] Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.