Maseeh College of Engineering

Project Proposal

Submission 1.0

A Block of Code

Senior Capstone Project

Team Members:
Nathan Bryant
Daniel Frister
Tyler Hart
Jacob Micikiewicz
Greg Stromire

Erebus Labs:
Dr. Mike Borowczak
University of Wyoming
DR. Andrea Burrows
PSU Advisor:
Roy Kravitz

February 27, 2015

Contents

1 Needs Statement **2**

2 Objective **2**

3 Background **4**

4 Marketing Requirements **4**

5 Current Proposed Solutions **5**
5.1 System Overview and User Interface 5
5.2 Blocks And Communication: Software 7
5.2.1 Determining Location 7
5.3 Language, Parsing, Processing 10
5.3.1 Lexing 10
5.3.2 Parsing 10
5.3.3 Interpretation 10
5.4 Visual Feedback 12
5.4.1 I/O 12
5.5 Micro-controller 13
5.6 Physical Connections 15

6 Risk Management **16**
6.1 User Interface 16
6.2 Software 16
6.4 Visual Feedback 17
6.5 Micro-controller 18
6.6 Physical Connections 18
6.7 General Cautions and Considerations 19
6.7.1 Fetching Block Tokens 19
6.7.2 Open Sourcing 20

1 Needs Statement

Today's world is one of interconnected technologies, at the heart of which are devices like computers and micro-controllers. As the technological complexity of our world increases we will find our society increasingly reliant on the men and women who know how to operate and control these devices.

However as of today no universally accessible curriculum includes a focus on the knowledge and skills needed to program these machines. We have need of any system that can help children learn and understand these concepts. Additionally, the long term effectiveness of any system will depend on both it's ease of use and it's accessibility across a large range communities, especially those under financial duress.

2 Objective

The primary goal of this project is to create a low cost tactile teaching aide capable of helping a wide range of K-12 students learn the basics of computer programming. Our proposed teaching aide is a set of blocks that will allow students to perform basic programming operations and assignments through the arrangement of the blocks. Verification of valid code structures will provide feedback for learning, and control of simple outputs connected to the system can give students a goal for their projects.

Project Requirements Sheet

Sponsor Requirements	Engineering Requirements	Justification
2	Device must use open source software.	Project should be expandable by others after the team finishes.
6	Device must have low power consumption.	Circuits will be enclosed inside a sealed shell, and should have a long enough operation on a single charge for a class session.
7,10	Device must be easily portable.	Target audience for the product is young to middle aged children.
7,10	Power will be provided by external power or rechargeable battery pack.	Device parts will be enclosed inside blocks, and user will not have to open any block to operate device.
8,10	Block function selection must be clearly visible.	Users cannot properly operate or build programs without knowing what each block represents.
10	Each block must be able to indicate proper operation and placement.	Users will need feedback while developing code to learn from their mistakes.
4,9	Essential programming elements must be represented in system by a block.	Functional programs require standard programing elements
4,9,10	Must compile and run simple programs.	Creating programs is the main function of the system, and necessary for instruction.
10	Device must produce an output based on the program compiled.	Users must have a useable/knowable result.
1,2,3	Circuits must be built from common components.	Custom ordered parts raise unit price and prevent product from being rebuilt without redesign.
1	Part selection for devices will be aimed at extended prices.	Final product will involve a larger number of blocks, and extended prices will be a better representation of actual production costs.

Sponsor Requirements

1	Low hardware production costs.
2	Open Source Hardware Design & Board (can use “closed source” components: ASICs, uC, etc.)
3	Must Have a Multi-Chip solution – e.g. no single SoC;
4	Fundamental grammar functioning, assignments, and binary operators.
5	Open software repository.
6	Low power operation.
7	Built in power source or power pack.
8	Blocks should have multiple possible functions.
9	Control structure blocks.
10	System must be accesible to novice coders, and is intended for use in a class room, by children of approximatley 10-14 years of age.

3 Background

Currently there are limited methods of teaching younger groups of students about computer programming, and those that do exist often rely heavily on access to computers. This project is meant to produce a learning aid that will function in a classroom setting, doing away with the need to have computers present for learning about programming. The use of a visual-tactile system to teach new skills to children has a long and well established history dating at least as far back as Froebel, who believed that freedom to play and experiment, was essential to real learning.

In terms of this project the system can be thought of as a, Froebel *Spielgaben*, or a subject specific learning module. The purpose of which, is to allow students of any age to be introduced to coding, using a familiar process that promotes creativity. Allowing the module to be untethered, separate from a computer, will also allow the teaching of programming without the other distractions available on a PC, and give students that have trouble with abstraction a set of objects to focus on.

4 Marketing Requirements

The final package will be a set of between twenty to thirty blocks typically two or less inches across a side. The set will function as it's own interface device, where the topographical arrangement of elements (blocks) will determine the *program*. The individual blocks must be identifiable as belonging to a specific programming construct-group. At a minimum these groups must include; numbers, variables, operators, and controls.

Additionally , if possible, a block should be capable of indicating what value or function has been assigned to it. Blocks should be easy to assemble, allowing users to experiment with layouts, but provide area specific error feedback, thereby increasing understanding and limiting confusion.

Finally, the set should be open source and be sufficiently accessible, both by cost and teaching use, as to promote universal access and encourage development and expansion by others.

5 Current Proposed Solutions

The current solutions we propose below are subject to change, as feasibility and effectiveness will become more clear as prototyping commences. Each level of description is limited to an overview of the specific proposed method. Each subsection is intended to create enough familiarity with the method described so that we can address associated *Risk Management* in section 6.

5.1 System Overview and User Interface

Our product will consist of identical *blocks*, each with enough digital circuitry to accomplish the following;

1. Electrically connect with each neighboring block,up to 4 blocks.
2. Connect to a global buss, carrying system power and ground.
3. Communicate over the global bus.
4. Obtain or store an objective value (its token ID) for the block
5. Control a single LED.

The block-to-block metal nets will form a mesh grid (local communication), while the global bus will use an I2C(or similar) protocol. We will address this network as a *matrix* for the rest of this document.

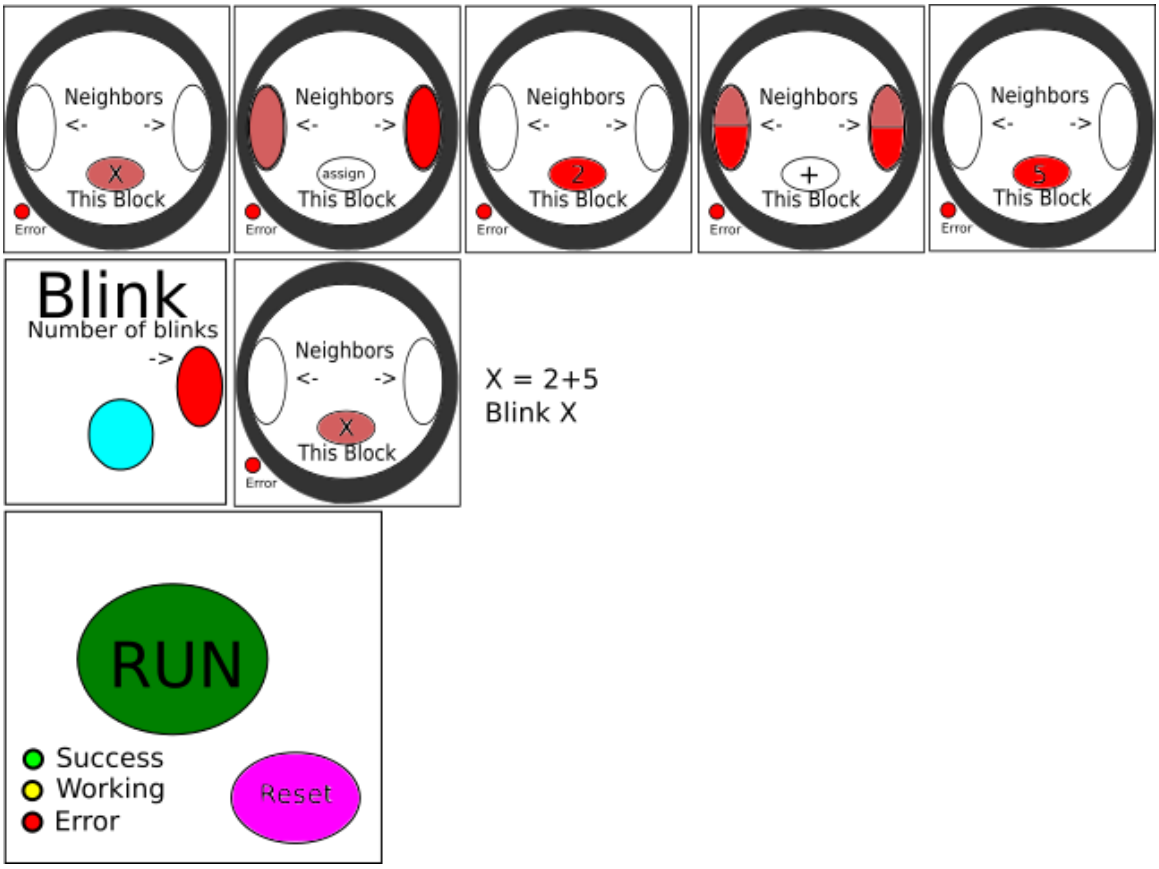


Figure 1: Visual aid for section 5.1

The ***RUN*** block, or control block (see figure 1), has buttons to force a reset or to attempt a run/ compile. It also has LED indicators for relaying current status. It connects to the bottom most left most of the other blocks.

`<Note: the bottom is one possible configuration being considered>`

Each programming block connects to its neighbors to form a program. The lines of code are read from left to right, and lines are read one at a time from top to bottom. On each programming block a tunable wheel allows the user to select the meaning or *value* of the block in its ***This Block*** window. Users are given hints for which blocks ought to be placed to the left and right of the current block. Hints appear in the ***Neighbors*** windows, and an LED will indicate error locality at the block whose sequence/ syntax is invalid when the program is run.

Output blocks will have some way of acting on the world, such as a controllable light source, or a small speaker for sound. The interface to control output is a separately connected special *output* block (not shown), which may be connected to the run/ control block.

5.2 Blocks And Communication: Software

Each block stores its own associated data. Each block stores a *position vector*, which represents where on the matrix the block is located, and stores a *unique ID*, which allows the block to be uniquely addressed on the global bus. Upon initialization, the position vector of each block is unknown.

5.2.1 Determining Location

The process for each block to identify its position vector is as follows:

1. First the positions of each block in column 1 are discovered:
   (a) The processing block passes a

   `<0,0>`

   vector to its northern neighbor via the local mesh grid.

   (b) Until the end of the column is reached, whenever a block receives a vector from its southern neighbor, each block increments the y component of its position vector, emits a message on the global bus containing

   `[position vector, unique ID]`

   then passes its position vector to its northern neighbor.
   $**$ *note*; *the unique ID is generated and assigned by I2C*

   (c) The last block on the column increments the y component of its position vector, then emits a message on the global bus containing

   `[position vector, unique ID, END_OF_DIMENSION]`

2. The processing block collects this info in its own local copy of the matrix.

3. To identify each block in each row:

   (a) The processing block iterates over each Column 1 entry in its local-cache. Using the emitted Unique IDs for each entry, the processing block sends a

   `BEGIN_RASTER`

   message to each Column 1 block via the global bus.

   (b) When each Column 1 block receives a

   `BEGIN_RASTER`

   message on the global bus, it passes its Position Vector to its eastern neighbor via the local bus.

   (c) Until the end of the line is reached, whenever a block receives a vector from its western neighbor, each block increments the x component of its position vector, emits a message on the global bus containing

   `[position vector, unique ID]`

   then passes its position vector to its eastern neighbor.

   (d) The last block on the line increments the x component of its position vector, then emits a message on the global bus containing

   `[position vector, unique ID, END_OF_DIMENSION]`

4. Using the messages passed on the global bus, the processing block is able to fill in its local-cache with all of the Unique IDs of the blocks in the network.

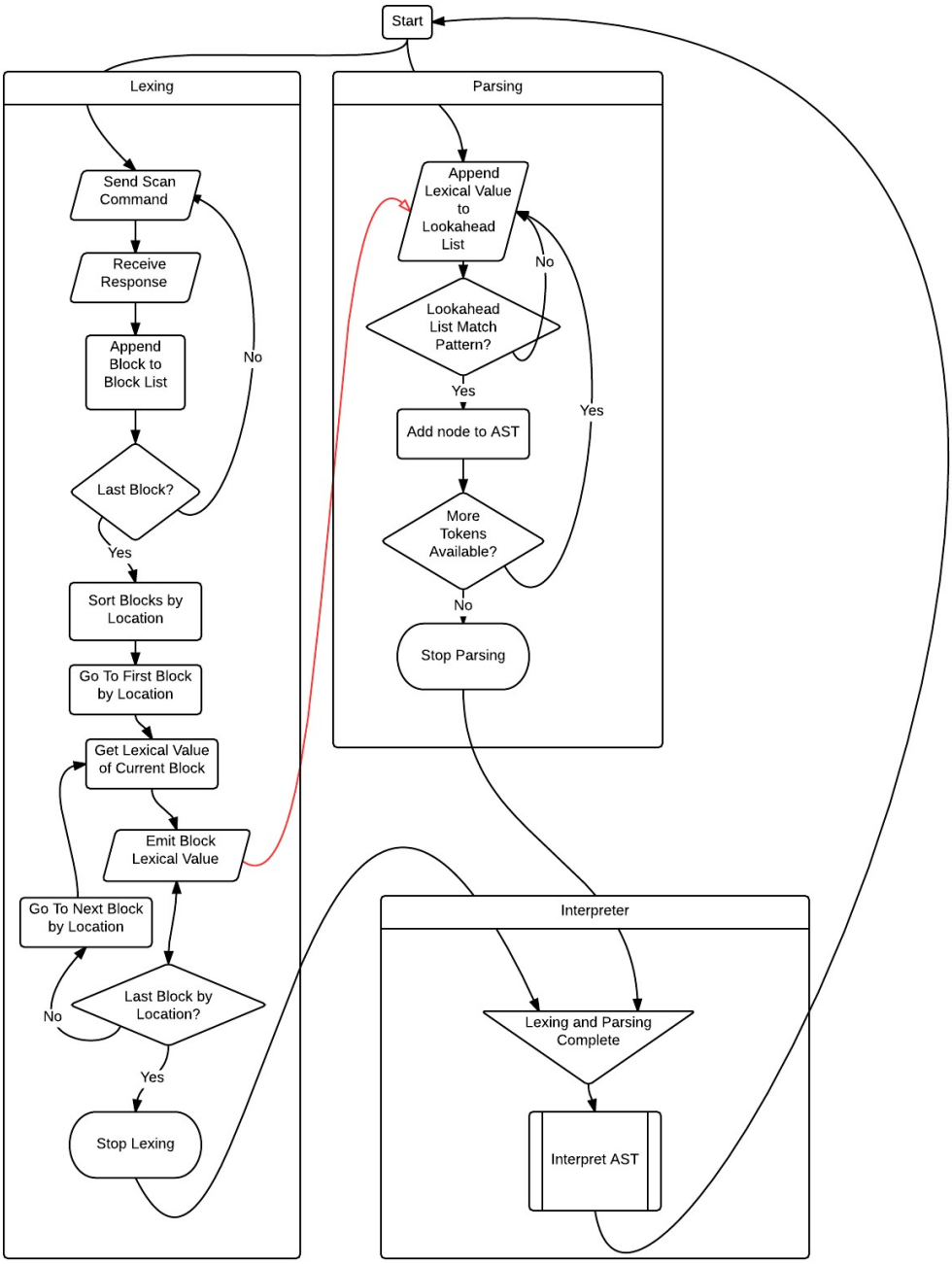


Figure 2: Flow chart for sections 5.2 through 5.3

5.3 Language, Parsing, Processing

After filling in its local-cache, the processing block can interpret the language in a traditional manner using a Lexer, Parser, and AST-Interpreter. This all happens in software.

5.3.1 Lexing

The processor is coded with an alterable lookup table containing the token associated with each Unique ID. The lexer does a raster scan on the processors local-cache matrix, and for each Unique ID it encounters, looks up the lexical value of the ID in the lookup table. The lexer hands this token, the value found (*see section 5.1 item 1 and section 5.4*) to the Parser.

5.3.2 Parsing

The parser is a traditional LR parser that constructs an abstract syntax tree from the lexical tokens handed to it by the lexer. It passes the root of the abstract syntax tree to the interpreter.

5.3.3 Interpretation

The interpreter descends the abstract syntax tree ***(AST)***, performing the actions associated with each node.

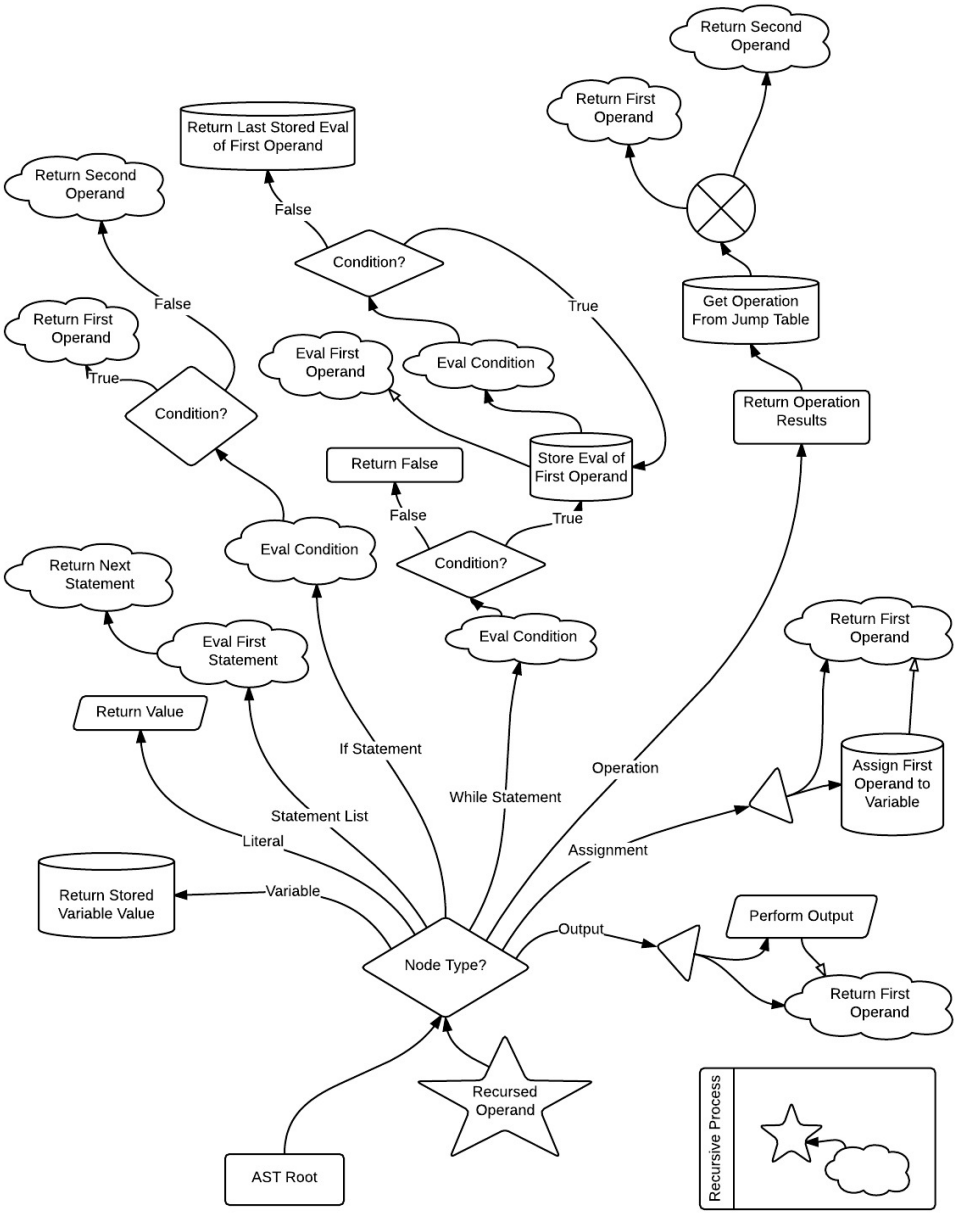


Figure 3: Flow chart for 5.3.3

5.4 Visual Feedback

Currently the scheme for allowing the user to set the block token (see 5.3.1) to be handed to the parser involves a system of selectable visual cues on the top face of the block. A potentiometer will be attached to a coded wheel that sits under the top face of the block, with a radius such that just the edge of the wheel may be manipulated by the user through a cut out on the side face of the block (see figure 4 below).

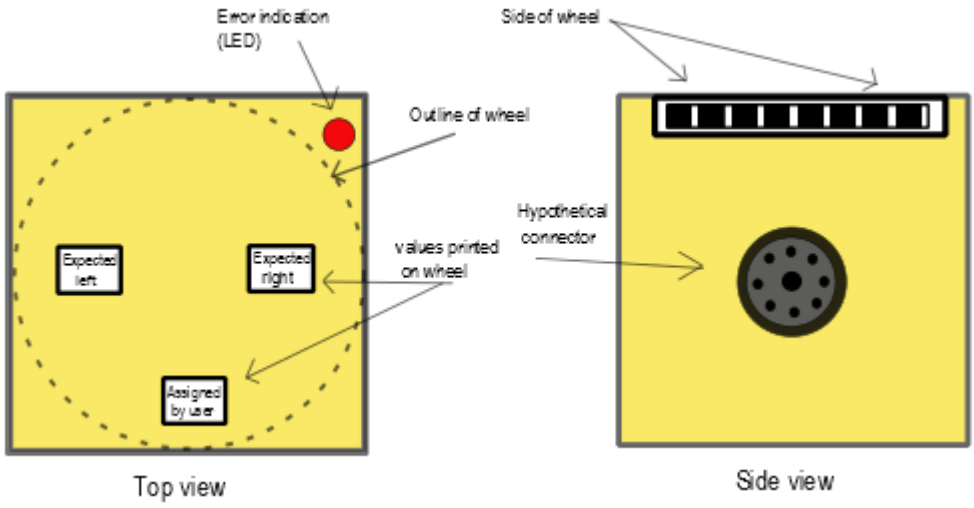


Figure 4: Wheel Encoder Diagrams for section 5.4

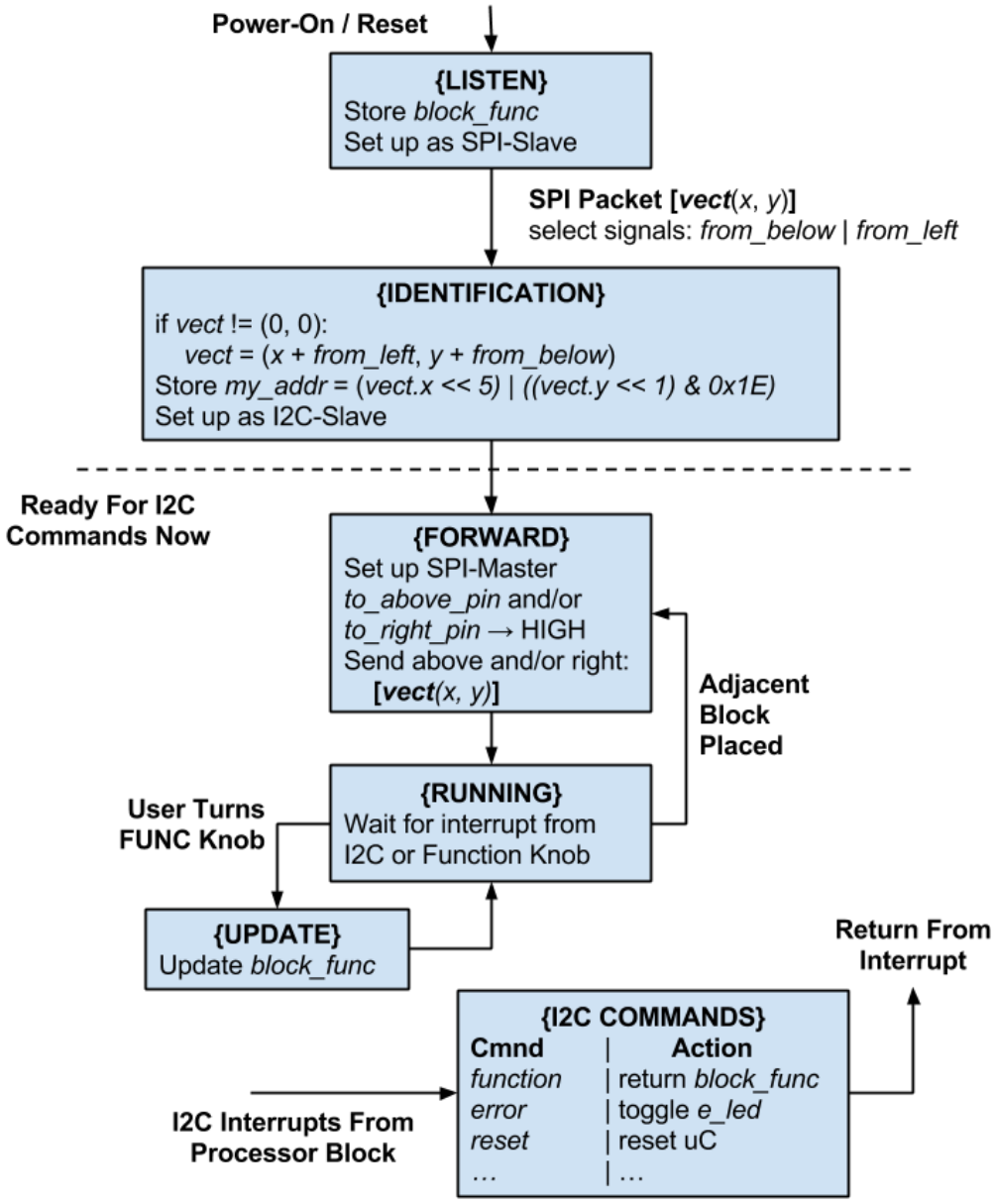
5.4.1 I/O

User interface by rotating the wheel to select the desired value (token), this corresponds to a number processed via Grey Code or ADC which is then sent to the lexer for lookup.

When the interpreter has finished compiling the program image, if any errors exist, the control unit identifies which unique ID holds the token that caused it and sends out on the bus;

```
[unique ID]
FLASH_LED
```

5.5 Micro-controller



Once a micro-controller has determined its address, it will enable its I2C port and attach itself to the shared bus. This allows the main processor SBC to regularly poll the bus for the correct devices.

<Note: I2C is one possible protocol being considered>

The diagram on the previous page assumes that the main processor **block** is oriented at the bottom-left of the code blocks and the **program** will be built up from this origin, i.e. each block receives its position vector either from below or from the left.

This implementation allows us to utilize existing, supported, documented I2C drivers for both the micro controller in a hardware-supported and interrupt-driven slave-mode and the SBC in master-mode.

If the block is disconnected and reconnected it should return to the beginning state. This is now noted at the top by Reset.

In a 2x2 example, the block at (1, 1) would get vector signals both from the left and from below. It could simply use the first one it receives and ignore the other, or it could use the second to confirm the correct position as they should generate the same coordinate value.

Each micro-controller generates its own address from its position vector. A possible implementation is that the x-coordinate is the most significant 3 bits of the address, followed by the y-coordinate as the next 4 bits of the address, and the final bit (LSB) is the I2C R/W bit, per spec. For example, the block at position (2, 1) would have the address of: [010 0001 x]. This resolves to dimension limitations of 8 blocks long and 16 blocks tall, or a max 16 lines of code that are max 8 blocks wide. This is further limited by I2C reserved addresses. Other implementations could improve upon this.

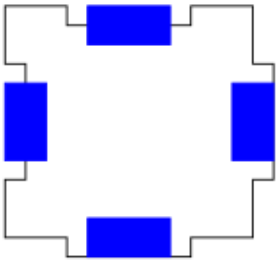
Once the block knows its function and its position, it is immediately available for I2C addressing from the SBC. Since the I2C communication will be interrupt-driven, the block can resume setting up after it services a message if it was received before it completed initialization. Possible commands could include:

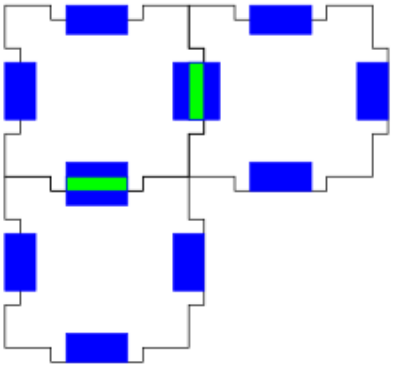
```
* Return this block's function code
* Toggle this block's error LED
* Reset this block
* Etc.
```

5.6 Physical Connections

Blocks will be interlocked using small outward pegs with corresponding recessed cavities. These protrusions on the blocks will contain the electrical connectors, and serve as a physical guide to ensure the pin connections line up during product use. A combination of varying the connector types, the protrusion shapes, and magnets can be used to hold blocks firmly together and will ensure that only the correct face of one block can be mated to a second block.

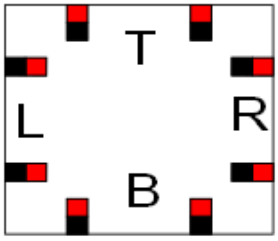
Top and left sides are male connectors

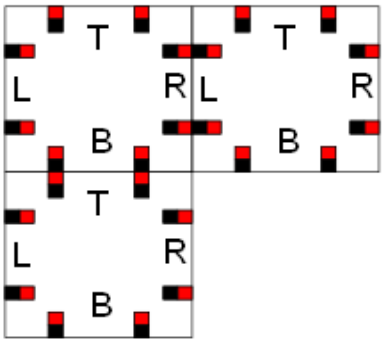




Bottom and right side are female connectors.

Top and right connectorscare north pole outward.





Bottom and left are south pole outward.

With the magnets, block pegs, and connectors in place each face will have a unique combination to ensure which connections are made.

6 Risk Management

Identifying potential risks associated with our current proposed solutions, can help by allowing a certain amount of mitigating action to be built in to the design process. This section follows the same layout as section 5, with 6.1 being directly related to 5.1 and so on.

6.1 User Interface

In any scheme designed to be easy to use there is the danger of being easy only for the designers. What we think of as intuitive is not certain to be easy to understand for students or teachers. The more complex our manuals or instructions are, the clearer it can be how to use the blocks.

There is however, a limit to how much time a user is willing invest in reading instruction before they can use the blocks effectively . If there is not enough or too much user targeted documentation, learning to use the blocks could be frustrating.

1. *Mitigation:* Both risks can be reduced by testing, and seeking feedback, which invites a trade off of time and resources spent on testing versus spent on designs and implementations.

6.2 Software

Possible pitfalls for the current proposed solution are discussed in this section, and likely mitigation techniques are discussed for each pitfall.

1. Varying path lengths on each bus may result in bit drift and/or clock skew.
   (a) *Mitigation:* Throughput on buses is not likely to be a significant factor in system performance (whereas interpreter performance is), so slower baud rates are a cheap solution.
2. A user can write an infinite loop

(a) *Mitigation:*User programs will run in their own process which will be killed whenever blocks are rearranged (then the new program will start in a new process).

3. Manufacturing uncertainties: Connectors do not align; enclosures are not square. Magnetic interference.

   (a) *Mitigation:* Expose connectors for proof-of-concept revisions to demonstrate functionality.

4. Communication algorithm was inherently flawed.

   (a) *Mitigation:* We plan to release our first revision at the halfway mark before "product launch". Reimplementing the Location-Vectoring algorithm using a tray as a fixed-grid reader of Unique IDs would be trivial from a design perspective.

6.4 Visual Feedback

In general there is a proportional relationship between the level of detail relayed to the user and complexity. The current proposal involves several moving pieces which each represent a risk of mechanical failure. Additionally there is a hardware to software interface at each point that posses additional risks. By contrast limiting the feedback system runs the risk of making the program controls too difficult.

1. *Mitigation:* Use of an ID *stub* in software will help simplify testing and allow for bugs to be narrowed to either HW or SW.

2. *Mitigation:* We will remain willing to limit the number of selectable states to reduce complexity as needed.

3. *Mitigation:* First run/ prototypes may also have to use a simple dial without the encoded wheel as proof of concept.

6.5 Micro-controller

Blocks may get stuck in different states.

1. *Mitigation:* Reset triggers and watchdog timers can help alleviate some issues.

Additionally the algorithm on its own does not determine end of line or completed search, the main processor block must poll regularly for block identities. This can lead to:

```
* Stale function values
* Wasted processing on both ends of the bus
```

2. *Mitigation:* Possible optimizations can be made by limiting poll to likely addresses (blocks starting at 0, etc)

6.6 Physical Connections

The interlocking of the blocks adds the possibility that blocks can only be inserted easily on faces that have no adjacent blocks already connected. This would mean that if the interlocking sections of blocks are tight tolerance, rows will have to be constructed prior to being connected to other rows.

Additionally adding a magnetic connection to hold the blocks together makes assembly of the block more difficult. Having unique locations for each connector/interlock pair means that an incorrectly oriented magnetic internal to the device could prevent a good connection.

1. *Mitigation:* Determining the tolerance between blocks will aid in the selection of an appropriate connector.
2. *Mitigation:* Use male/ female type connectors only for first pass, avoiding magnets.

6.7 General Cautions and Considerations

6.7.1 Fetching Block Tokens

When the token is selected on a block by the user(*see 5.3 and 5.4*) it must be fetched by the lexer. Several solutions exist and are proposed below. The solution chosen is dependent on metrics obtained during prototyping. The following factors will be weighed when determining which block-fetch technique will be used:

1. *Required communication transactions.* Some techniques include the block token in messages already required by the location-finding algorithm, while some introduce additional messages.
2. *Token-Fetch-to-Parse Latency.* The amount of time it takes from reading the user-selected token to passing the token to the lexer is variable from technique-to-technique. This latency is a component of the overall system opacity time, eg. the time when user-changes to selected block tokens don't effect program execution. Opacity time should be reduced where possible.
3. *Determinism.* Some techniques use parallel execution which could lead to certain token-fetches to arrive at indeterminate times. This can make debugging during testing phases more difficult.
4. *Complexity.* The complexity of the algorithm chosen and the complexity of implementing the algorithm are to be considered. Algorithmic complexity makes later expansion more difficult, while implementation complexity increases development time.

The following block-token-fetching techniques will be considered, and their consideration of the above factors are included:

1. *Reading and sending block-tokens during location-scanning time.* Least amount of communication, most token-fetch-to-parse latency, deterministic, least complexity.
2. *Fetch during lex time.* Adds a message transaction for each block, minimum token-fetch-to-parse latency, deterministic, slightly more complex.

3. *Regular polling in asynchronus thread from lexing.* Increases the amount of messages by indeterminate amount, indeterminate token-fetch-to-parse latency, non-deterministic, complex implementation but least complex algorithmically.

6.7.2 Open Sourcing

To stay on track with developing an open source platform, we want to reduce actions that might lead to complications.

1. *Mitigation:* avoid dependence on non-generic components
2. *Mitigation:* avoid, where possible, proprietary technologies.
3. *Mitigation:*always check licensing