

Logic Tiles

A Senior Project

presented to

the Faculty of the Computer Engineering Department
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science, Computer Engineering

by

Tristan Lennertz & Andrew Wheeler

June 2017

Table of Contents

| | |
|-------------------------------------|-----------|
| Table of Contents | 1 |
| Introduction | 2 |
| Overview | 2 |
| Background /Design Decisions | 2 |
| Project Goals | 3 |
| End Users | 3 |
| Design Requirements | 4 |
| Engineering Requirements | 4 |
| Hardware Design | 5 |
| Tile Design | 5 |
| Triangular Tiles | 5 |
| Rectangular Tiles | 5 |
| Subject Testing and Decision | 6 |
| Board / Module Design | 14 |
| Software Design | 19 |
| Data Structures | 19 |
| State Machine | 20 |
| Circuit Traversal Algorithm | 21 |
| Implementation / Results | 22 |
| Testing | 22 |
| Tile Detection | 22 |
| Circuit Creation | 22 |
| Results | 23 |
| Further Implementation / Next Steps | 23 |
| Conclusion | 23 |
| Works Cited | 24 |

Introduction

Overview

This project was originally conceived by Professor Andrew Danowitz as he considered the restructuring of the introductory digital design course at Cal Poly. As it stands now, students apply their knowledge of boolean algebra and combinatorial logic through the programming of a Field Programmable Gate Array (FPGA) using a Hardware Descriptive Language (HDL). While this is the industry standard for designing large, complex digital circuits, and is an fundamental skill to learn, there is a lack of actual circuit building in the process that can cause a disconnect from theory to application for students who have little-to-no experience with digital logic. Our project attempts to bridge that gap by providing a physical device with which students can manually develop their own digital circuits, forming them on a board as they place representative tiles onto it. They can then view in real time how their circuits are affected by certain inputs or restructuring of connections. This would be a useful supplement in the course that helps to emphasize the descriptive, and not procedural, nature of HDLs.

Background / Design Decisions

To begin formulating the design of our project, we began by researching similar methods employed to teach programming to children and young adults. The closest thing we found to our notion of the project was a children's toy called Cubetto; Cubetto is a small robot that is controlled by a board that children place passive tiles onto in order to issue instructions. The concept of this board was very similar to the board we had envisioned, and so we did more research into Cubetto's inception. This area, called *tangible learning*, eventually led us to a thesis by Timothy McNerney of MIT in 1983, in which he proposed physical building blocks that were used to construct simple programs (his work would later lead to the development of the Lego Mindstorm series of educational toys) [2]. His application of the idea, however, was through the process of stacking many active units that would communicate via I²C, which did not meet our desire to utilize passive tiles like Cubetto. McNerney's work, however, did lead us to an article by Andrew C Smith of the Meraka Institute who utilized passive elements on a board to manipulate a toy Robot [4]. From his research we discovered that the passive elements

contained a series of magnets embedded into them that would trigger specific reed switches on the board itself, giving each block a unique code that could be deciphered by a connected microcontroller.

We decided to utilize this technique (reading magnets to decipher the passive elements on the board), but wanted to reduce the cost of the parts used. While Smith's design was simple and used the presence and absence of a magnetic field to determine the encoding, it made each passive element large and fairly expensive as it required up to 9 magnets to be placed on it. Furthermore, each place for a block on the board needed 9 reed switches, so Smith's prototype used a relatively large and costly play field. Conversely, our project needed a multitude of different tile types and many small spaces on the board to place them. From this we decided to use Hall Effect sensors instead of reed switches, as a hall effect sensor can detect a magnetic presence in the same way as a reed switch, but it can also differentiate the magnetic strength and polarity [6]. Because of this we would need to use less magnets on our passive tiles and would instead vary the strength and polarity of the magnet.

Before our final decision on the design, we had to determine if this magnet system was the best course of action, or if we should use our second idea for detection: RFID tags. Ultimately, however, we decided to use magnets for a several reasons. First, though RFID tags have gone down considerably in cost in recent years, it would still be more expensive to implement them over magnets. RFID tags themselves are relatively inexpensive (only a few cents per tag), but the actual readers are still too expensive for our purposes. The least expensive reader we found would cost about \$10 each in relatively low volume, and we would need at least one per tile space on the board. Compared to the hall-effect sensors (which cost <\$1 each, even at low volumes) the RFID reader's cost per tile area would be close to (or even greater than) three times the cost of the number of hall-effect sensors we would need for the same tile area. Another advantage of magnets over RFID is that we can deduce orientation of our tile from magnets. Since these tiles can be placed in different orientations, we need a way to discern this. RFID is unable to detect this, as it can only detect proximity, but we could gather the orientation from the polarity of the magnet.

Project Goals

End Users

As stated in our overview, the main end users of our project are Electrical and Computer Engineering students. While they are assumed to have a certain degree of technical experience, we should expect that a majority of them have little or no skills related to digital logic and design. They should, however, have the knowledge to manipulate a simple terminal interface and light debugging skills.

Furthermore, digital design professors would also utilize our system. They would most likely have some training with using our system beforehand would assist students with debugging their created circuits.

Design Requirements

Seeing as our end users are students with little experience, we need to make sure that our device is robust and easy to use. Adding a new tile or gate to circuit should be as easy as placing it on a corresponding place on the board and our system should quickly (within a second) detect that a new tile has been place and what the type the tile is. Students can then interact with the circuit using a companion terminal interface that is connected to the circuit board, allowing them to manipulate source values and read the signal value at various places on the circuit. Furthermore, we'd like our system to be able to report to the student when they have incorrectly constructed the circuit and give them a hint as to where the problem lies. We do not want to directly tell them where the issue is as this is an education tool and the act of debugging the circuit is a part of the educational experience. We also decided that, because of the availability of physical space in classrooms and the variability of scale in digital design projects, we would like our design to not be permanently fixed to specific physical dimensions, but instead instead allow for modularly sized boards. Finally, because many of these boards would needed in the classroom, our design should be inexpensive (relative to products of the same caliber such as FPGAs or children's programming toys) to produce.

Engineering Requirements

From our design requirements, we can produce a series of engineering requirements that can be used to determine the course of our development process. Our requirements are:

1. Control the detection of tiles and creation of digital logic circuit through the use of a microcontroller.
2. Within 1 second, detect the placement of a new tile and decode the strength and polarity of up to 3 magnets on said tile, determining its type and orientation.
3. Provide a terminal interface that takes in user input and communicated with the microcontroller using UART.
4. Keep costs of boards and parts ~\$200-250.

Hardware Design

Tile Design

We explored two options when choosing the basic design of the tiles: A traditional rectangular tile, or a more novel triangular tile. Because of our needs for encoding and recognition of orientation, a minimum of 3 sensors per tile is required for either shape. With that important, resource-intensive consideration taken off the table, it came down to two main factors:

1. Compactness of the baseline circuit (a 1-bit full adder)
2. Intuitive use for a student

The first requirement minimizes the required board size in tiles, and thus the cost, while the second fulfills the primary objective of this project. Our strategy was to independently design a system of tiles for either shape and test them with actual first-time digital logic students. Figures 1 and 2 show example layouts with mockup paper tiles of either system.

Triangular Tiles

A unique challenge for the triangular tile system was the requirement for at least 3 possible inputs to gates whose operations accept it. Associating each tile side with an input or output left the gate tiles one input short. In order to overcome this, wires on a tile became one of two types: normal, or offset (dashed/dotted lines). Offset wires could be reached by wire tiles that

converted normal wires to offsets. This meant that two signals could be bundled into one side of a tile, allowing for the a maximum of four possible inputs on each relevant gate.

Additional considerations include the fact that tiles have 3 possible orientations when placed on the tile board. This makes an encoding scheme and its software interpretation potentially more complex.

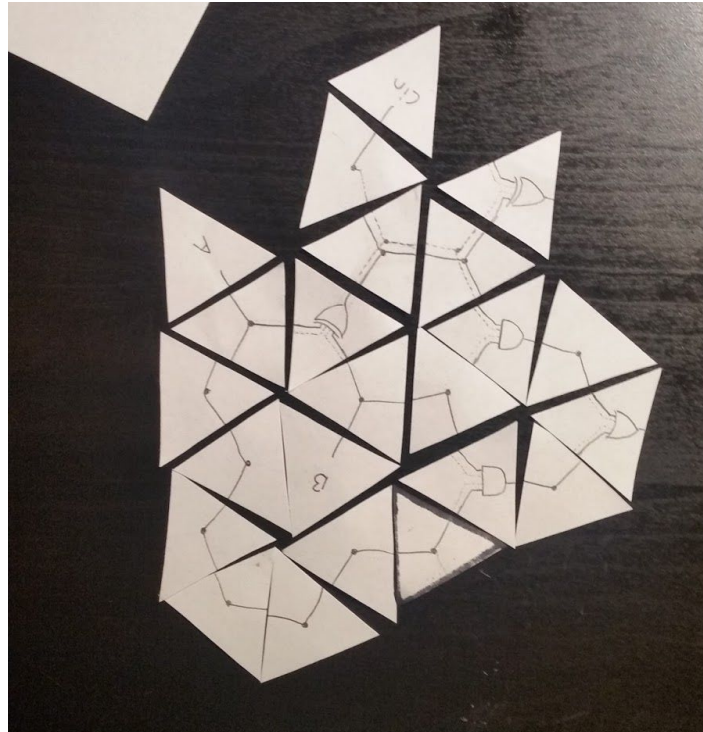


Figure 1: Paper Prototype of Triangular Tiles

Rectangular Tiles

The development of the rectangular tiles were done with parallelograms in order to emphasize the need to limit the number of possible orientations to two. Four potential orientations, such as with square tiles, would require an additional sensor or inordinately complex encoding and interpretation to properly identify each potential direction a gate or wire could be facing.

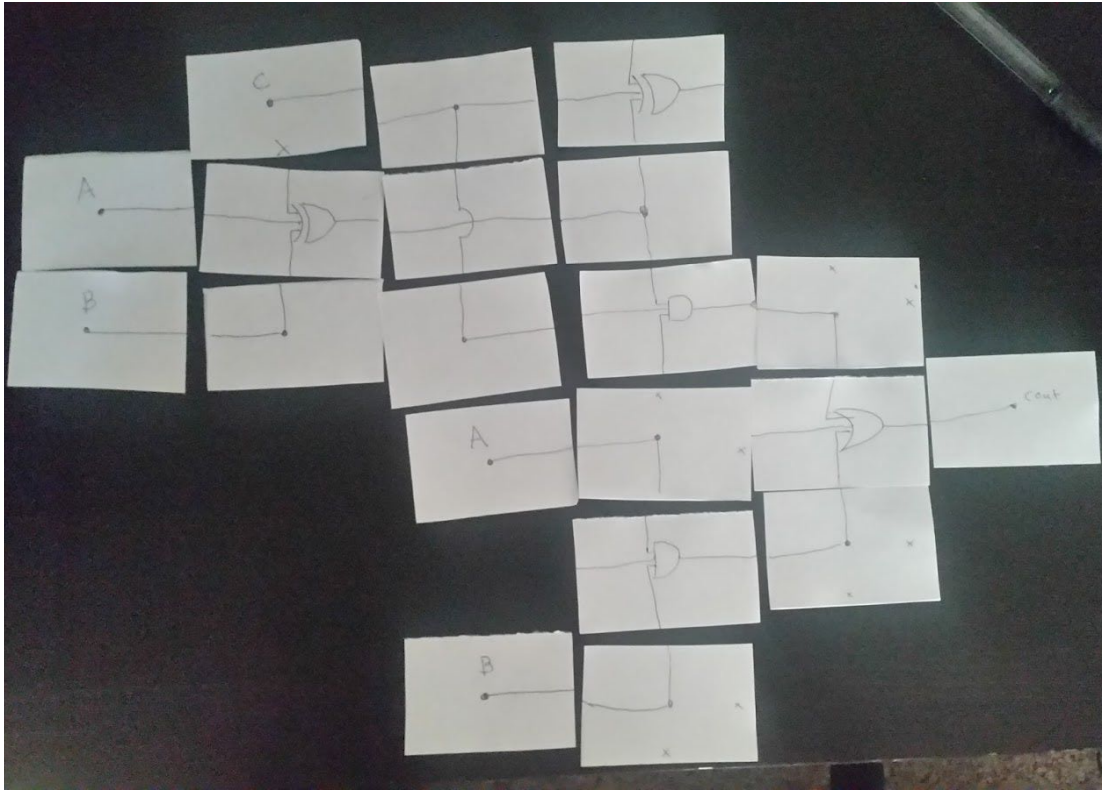


Figure 2: Paper Prototype of Rectangular Tiles

Subject Testing and Decision

We tested both designs using paper tiles of each shape with students who were newly introduced to digital logic in CPE 133. We gave them a reference circuit to construct and minimal prompting on how the tiles were meant to be put together. After observing each student work, and asking them about the experience after, we drew several qualitative conclusions:

1. A rectangular layout is much more intuitive than triangular tiles to begin to pick up and begin building circuits with.
2. Triangular tiles could build the reference circuit much more compactly, but it was difficult for a student to become aware of or take advantage of how to do so without additional hints and tips along the way.
3. Without the proper tiles provided in the correct quantity, a circuit constructed with the rectangular tiles could become unmanageably large quite quickly in order to route the wires around each other correctly.

We ultimately decided that the ability for a student to pick up the tiles and begin building a circuit with almost no additional help from an instructor was more valuable to us than having as small of a tile board as possible, and have gone with the rectangular tiles. That said, the triangular tile system is fascinating and quite a fun novelty to build with. The bundling of wires between a normal and offset plane with the triangular tiles also proved to be a feature that could be very useful in the rectangular tiles. While we did not implement this feature with our rectangular tiles, it is something that could be added as an improvement in future iterations.

As observed in our subject tests, a danger with our the rectangular tiles in particular is not having enough of them, or of the right type. This is not an issue if a large number of tiles can be cheaply supplied, however, which is one of the requirements we have set out and designed our physical tiles for.

The tiles are all 1.5" x 1" rectangles, with the design of the component each one represents raised up off the tile face. There are four separate hole locations on the underside of the tile, as shown in an example tile bottom in figure 3. The depth of each hole, as well as the polarity of the inserted magnet, determines the encoding value. The holes on either side of the tile (labelled mag0 and mag1) are used to provide a number of different encoding values to be detected simultaneously with separate hall-effect sensors on the board and used in combination with each other as identifiers for the type of tile being read. These encoding values are described in table 1 below.

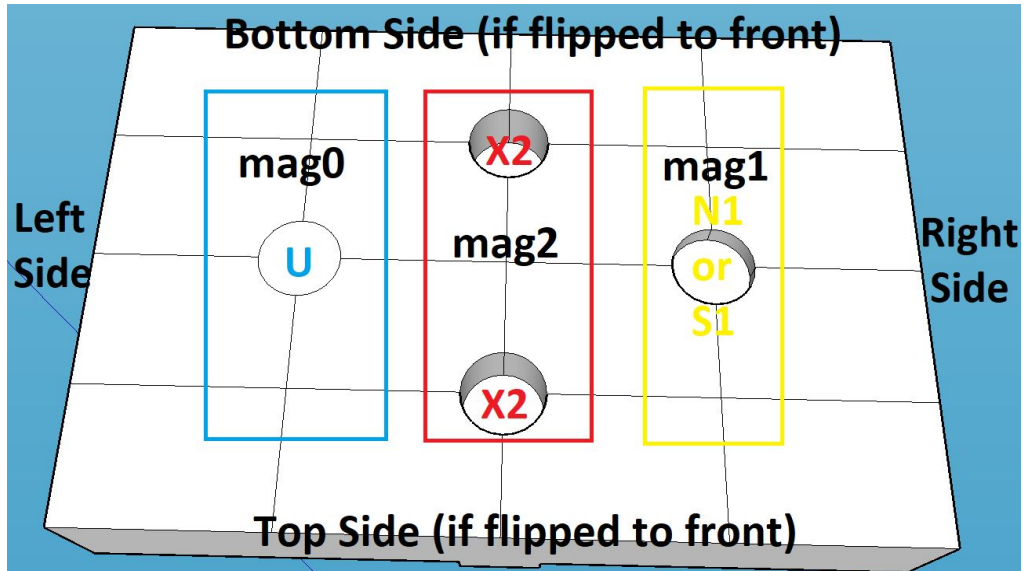


Figure 3: Example Bottom of a Tile

The central holes (labelled mag2) are used in conjunction with a single hall-effect sensor, located offset from the middle of the rectangular area of sensors for each tile location on the board. It's position is such that only one central hole and magnet will be above that sensor at any given time, with a flip in the tile's orientation putting the other central magnet above it to be read. This is the key to sensing the orientation. We have encoded the tiles such that the center holes on any given tile always have the same magnitude. These are the same magnitudes as N1/S1 and N2/S2 in table 1 below, and are designated as X1 and X2 when referring to them as encoding values to be used in combination with the side magnets. While the same in magnitude, they will have different polarities, with a *northern* polarity always being on the top central hole of the tiles, and a *southern* polarity always being on the bottom.

Table 1: Description of Tile Encoding Values

| Encoding Value | Tile Hole Depth | Description |
|----------------|-----------------|--|
| U | 0" | Simply the absence of a magnet, which is still a distinct range of values to be used as a valid encoding symbol. Is the quiescent voltage of the hall-effect sensor. |
| N1 | 0.045" | Just deep enough for the outward, <i>north</i> |

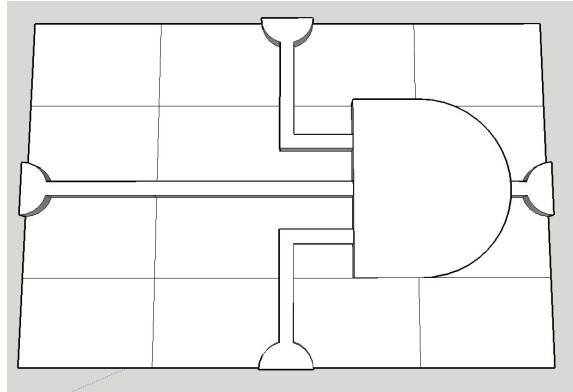
| | | |
|----|--------|--|
| | | polarity face of the magnet to be flush with the bottom of the tile. This is the closer, “stronger” northern value, which will have the highest positive value range for the sensor. |
| S1 | 0.045" | Same as N1, but with the <i>south</i> face flush with the bottom of the tile. Produces the lowest possible value range for the sensor. Note that while the magnets are 1mm in height, the hole depth is slightly larger to allow for 3D printing tolerances. |
| N2 | 0.125" | The deeper, “weaker” northern value (north side facing outward). This value range falls between U and N1. |
| S2 | 0.125" | The deeper, “weaker” southern value (north side facing outward). This value range falls between U and S1. |
| X1 | 0.045" | The larger magnitude for encoding the central magnets on a tile. <i>Either</i> N1 or S1. |
| X2 | 0.125" | The smaller magnitude for encoding the central magnets on a tile. <i>Either</i> N2 or S2. |

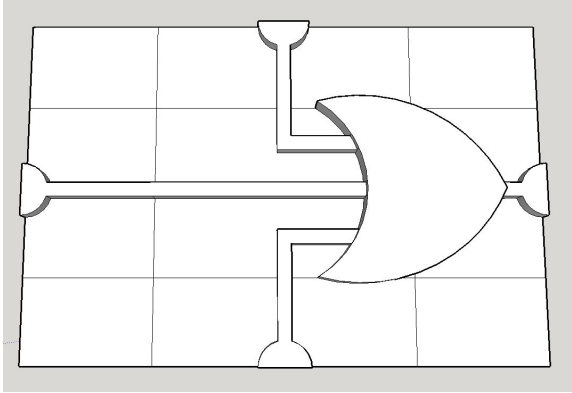
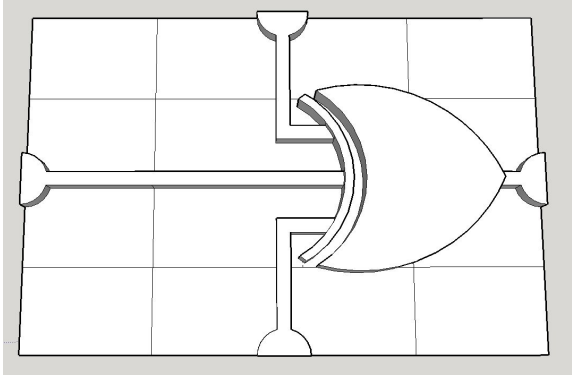
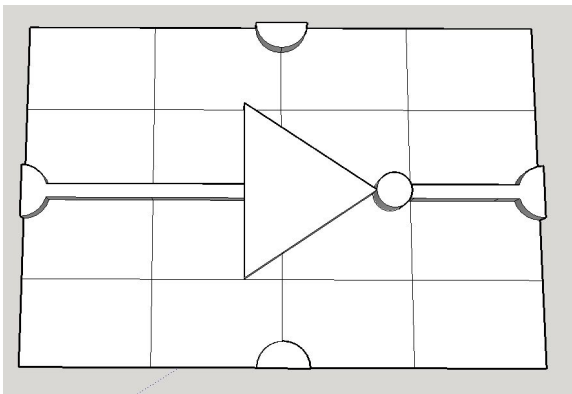
So, when reading the central hall-effect sensor for any given tile, we are able to tell whether it is a normal orientation (reading a northern polarity) or a flipped orientation (reading a southern polarity). Based on the orientation of the tile, we know whether to flip the read values from each side hall effect sensor (mag0 and mag1 in figure 3) into the encoding fields for the tile in question, swapping them if the orientation is flipped. With the encoding fields of the tile sides properly assigned, and a magnitude of the center determined, the tile's identity can be looked up, and the neighboring tile can be determined for each side.

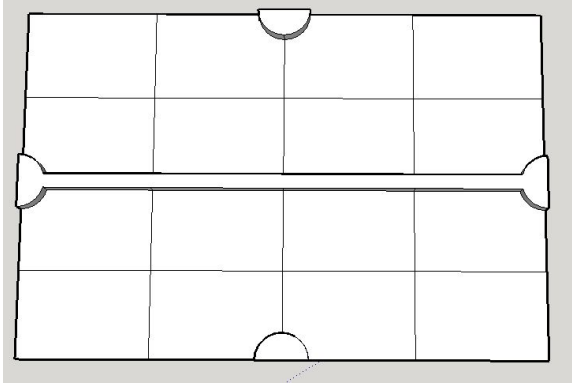
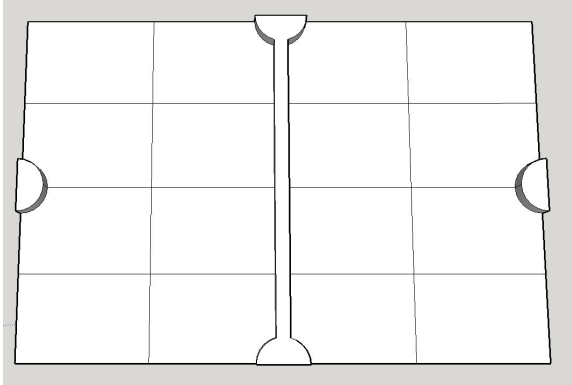
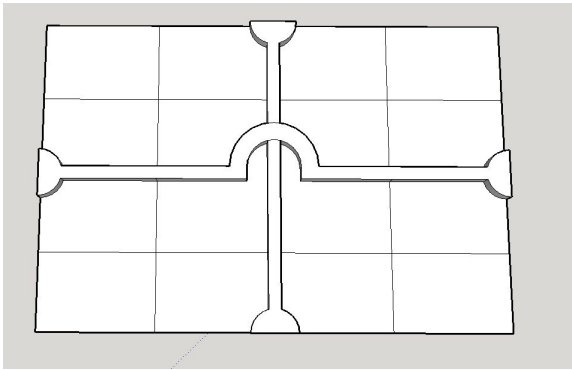
We designed each tile so that they can be 3D printed with relative ease, and then have their 3mm x 1mm neodymium magnets subsequently inserted. Our final tile designs follow in table 2 below, with their identifying encoding values included. Note that the wire types labelled with numbers use hands on a clock as their guidelines. Although the clock naming scheme makes for unambiguous tile names, some of them have their tops oriented such that the flipped orientation has the correct clock hand positions of the wires, and not the normal. This still does not create naming ambiguity, but might lead to confusion. For consistency the names have not been changed, but can be in future iterations.

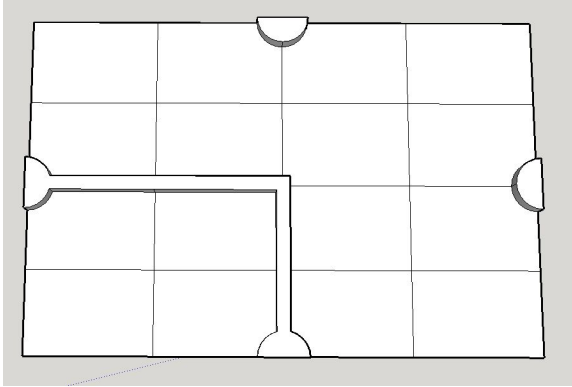
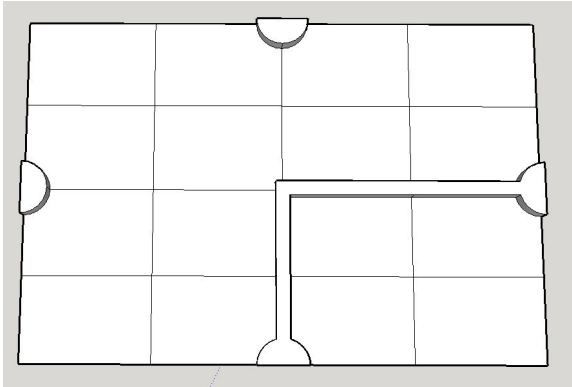
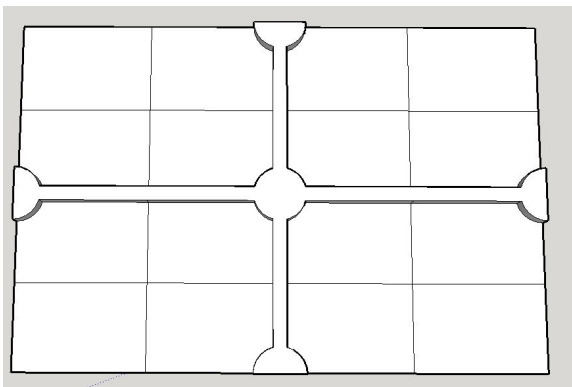
Additionally, while AND, OR, and XOR gates can take up to three inputs, two is still the minimum number required to be valid. Multiple source / probes tiles are also excluded from this table, as their designs are extremely similar. Their codes also follow incrementally from the first examples provided here, and can be referenced/modified in the provided code for this project if needed.

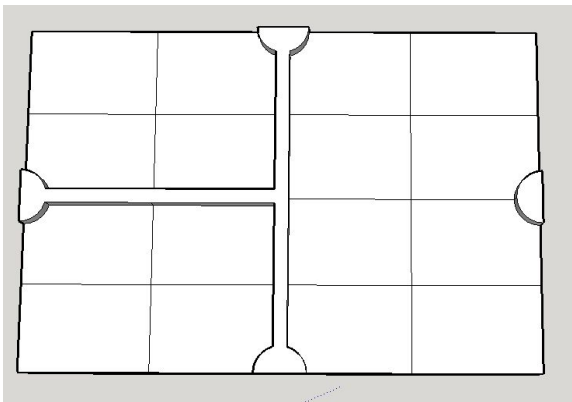
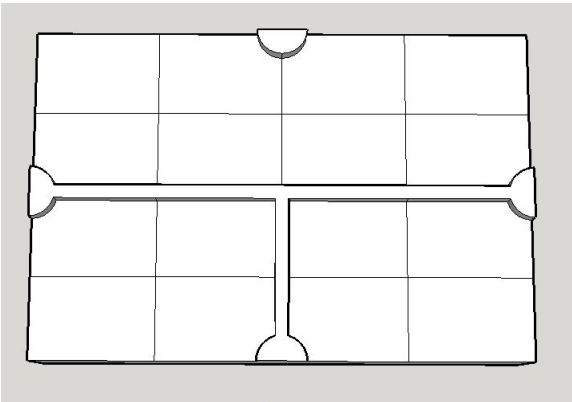
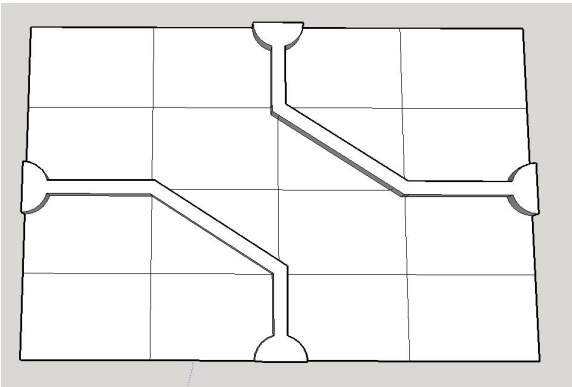
Table 2: Final Tile Designs and Encoding Values

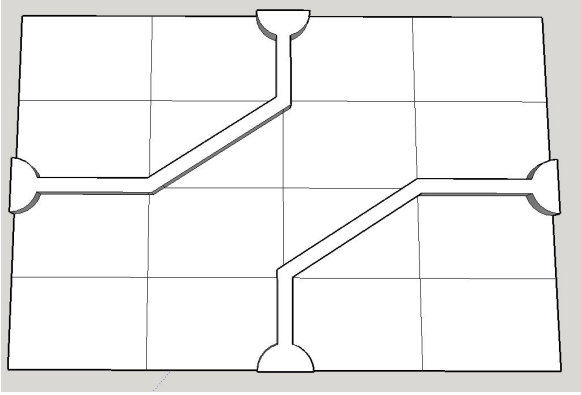
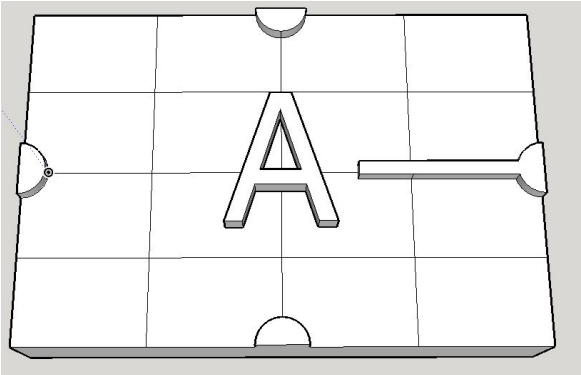
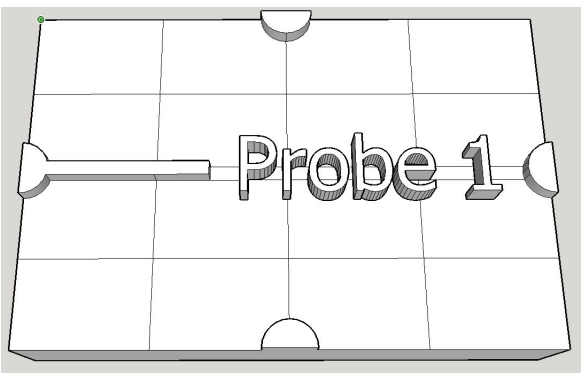
| Name | Design | mag0 | mag2 | mag1 |
|------|--|------|------|------|
| AND |  | U | X1 | N1 |

| | | | | |
|-----|--|---|----|----|
| OR |  | U | X1 | N2 |
| XOR |  | U | X1 | S1 |
| NOT |  | U | X1 | S2 |

| | | | | |
|------------|--|----|----|----|
| Horizontal |  | U | X2 | N1 |
| Vertical |  | U | X2 | S1 |
| Jump |  | N1 | X2 | N1 |

| | | | | |
|-----------|--|----|----|----|
| Wire_12_3 |  | U | X2 | S2 |
| Wire_9_12 |  | U | X2 | N2 |
| Ultranode |  | N1 | X2 | S1 |

| | | | | |
|------------------|--|----|----|----|
| Wire_6_9_12 |  | S1 | X2 | S2 |
| Wire_9_12_3 |  | N1 | X2 | N2 |
| Wire_12_3_Double |  | S1 | X2 | S1 |

| | | | | |
|------------------|--|----|----|----|
| Wire_9_12_Double |  | S1 | X2 | N1 |
| Source_A |  | N1 | X1 | N1 |
| Probe_1 |  | S1 | X1 | N1 |

These tiles are available in the project files as Sketchup projects to be modified, or .stl files to be 3D printed.

Board / Module Design

Our aim for the tile board was for it to have clusters of the three sensors, as described in the preceding section, below each area where a tile could be placed. We opted to mount these

clusters straight onto a printed circuit board in order to obtain the precise placements needed for consistent measurements across all tiles. Instead of designing a single, monolithic PCB to sense all of the tiles in the end product, we opted for a more modular design in which each PCB is a rectangular 4 x 2 tile board. These modules could then be grouped together in order to form larger boards as needed. There were several benefits to this:

1. Less initial investment in the design and development of a prototype board
2. More scalability in the size of the final tileboard. The intended final size for a tile board in order to achieve the baseline, 1-bit full adder circuit with ease is a 2 module x 3 module (8 tile x 6 tile) tile board. That said, the use of modular components mean there is no practical cap on what the size could be.
3. Development and testing done on a single module of 8 tiles is less cumbersome and helps avoid confusion in debugging
4. A dimension of 4 tiles x 2 tiles works very nicely in terms of software manipulation.

Besides the dimensions being a base of 2, the number of actual tiles, 8 (a byte), makes room for even more potential software tricks and numberwangs¹.

These modules, besides providing the backbone of the board and precise sensor placements, also serve to multiplex the outputs of each sensor together in a way that is easily accessible to the tile board's microcontroller. It is designed so that each module has only one analog output pin, and any magnet on the board can be selected for that output with the 5 selection pins, described in table 3 below.

Table 3: Module Pin Descriptions

| Pin(s) | |
|-----------------------|--|
| TILE_SEL0 - TILE_SEL2 | Selects which magnet cluster of the 8 on a module will be multiplexed to the output. SEL0 is the LSB, with 0x0 corresponding to tile 0 on the module, and 0x7 corresponding to tile 7. |
| MAG_SEL0 - MAG_SEL1 | Selects which magnet out of any of the clusters will be MUXed as the output. |

¹ <https://www.youtube.com/watch?v=qjOZtWZ56lc>

| | |
|-------|--|
| | <p>MAG_SEL0 is the LSB.</p> <p>NOTE: In our selected 3:1 MUX, 0x0 is a <i>high-impedance</i> output for the chip. 0x1 corresponds to mag0, 0x2 to mag1, and 0x3 to mag2.</p> |
| A_OUT | <p>The analog output of the module. This is the only signal that cannot be shared with the other modules. With our chosen sensors, should range between 0-2V, or high-impedance (if 0x0 selected on 3:1 MUX)</p> |
| VCC | <p>Input voltage can be 3.3-5V, as it does not affect the sensor output. 3.3V recommended, as that is the VCC of our selected microcontroller and uses less power.</p> |

Essentially, there are two stages of multiplexing on the board. First, all magnets of the same number (e.g. mag0, mag1, mag2) are multiplexed together on an 8:1 MUX. There are three of these MUXes – one for each magnet number – that take one magnet from each of the 8 tile clusters on the board. The three outputs of these MUXes are then fed into a 3:1 MUX, the output of which is the output of the module. When integrating multiple modules into a single board, for any given select pin on a module, that same pin on any other module will share the same signal line. That is, the number of GPIO pins on the microcontroller allocated to selecting tile magnets will always remain at 5.

The microcontroller will determine what selection pin values are appropriate for the given tile/magnet on the board it is trying to access, and apply those signals to all of the modules at once. In determining what select pins to use, it will also determine which module that tile is on, and will be able to read that desired module's output. The separate modules' outputs can either be multiplexed together, or simply attached to different ADC pins on the microcontroller. We

have opted to use the latter approach, but a good followup on this project would be to design an auxiliary board that serves to interconnect any given number of modules together in a way that is easy to assemble and access those modules with.

Figures 4 and 5 are from the module schematic, and show one of the sensor clusters and part of the MUXing scheme, respectively. Standard 0.1 μ F decoupling capacitors have been added for stability. A full image of the schematic is available in pdf form with the project files, as is the actual board layout. Eagle files are included for both as well.

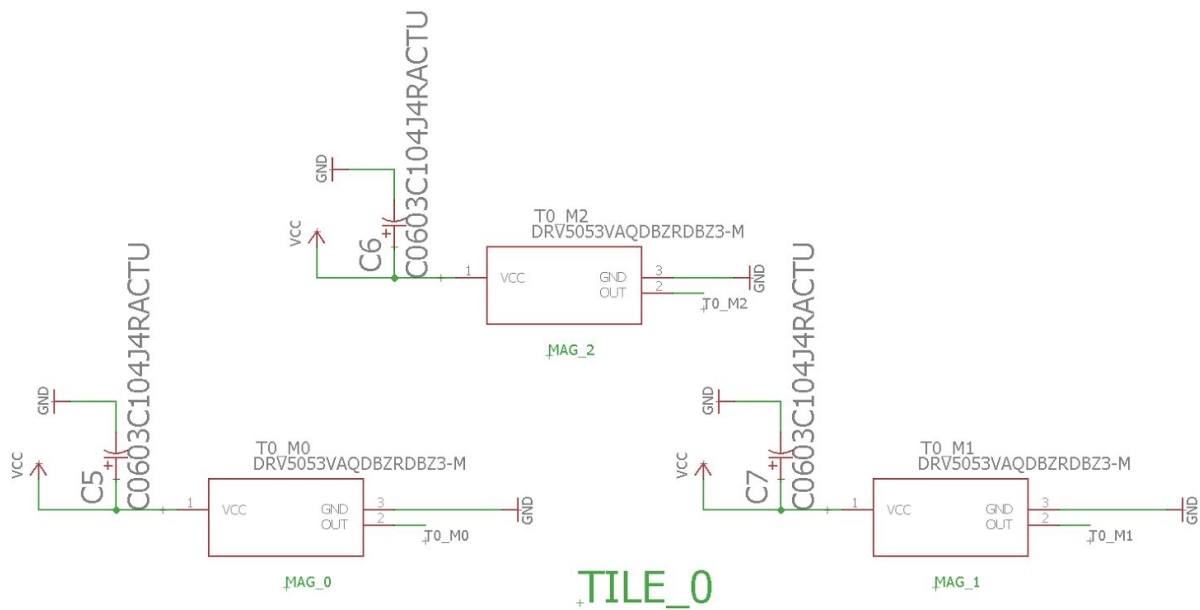


Figure 4: Sensor Cluster Schematic

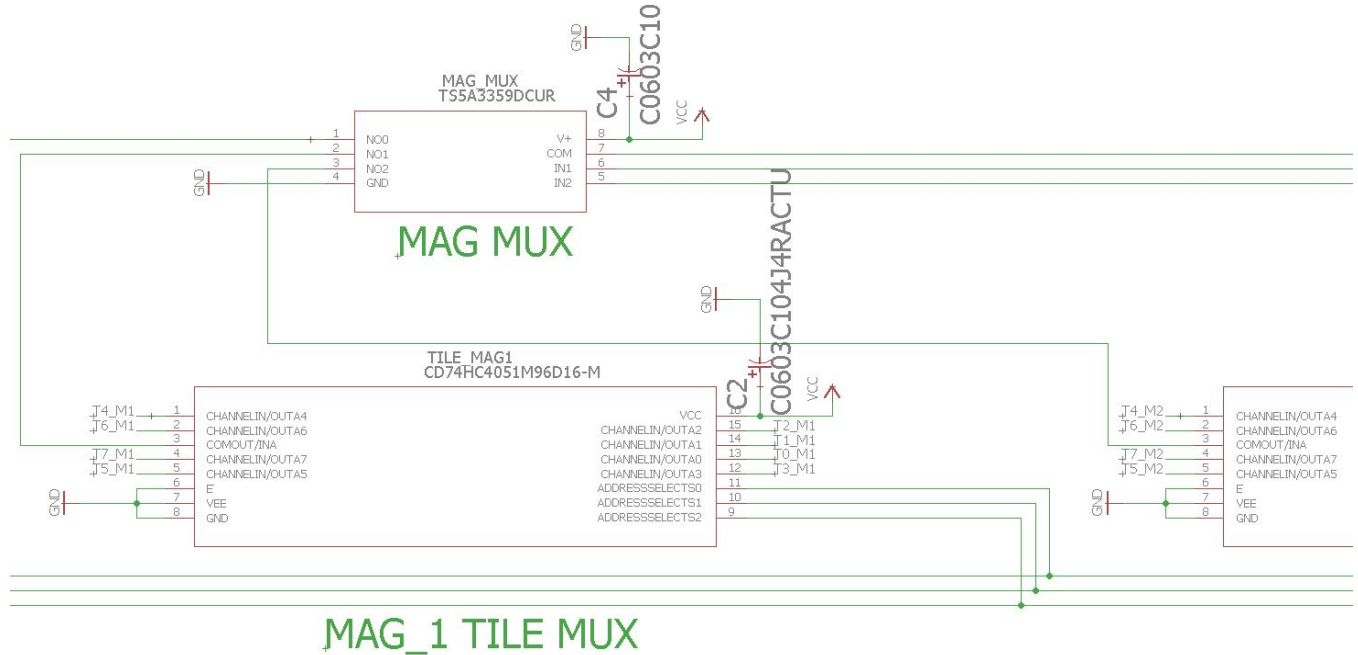


Figure 5: Partial Schematic of Module Multiplexing Scheme

The selected components' part numbers, maximum current draws (if applicable), and quantities per module are listed in table 4 below².

Table 4: Module Component Descriptions

| Component | Part Number | Quantity | Max Current Draw (mA) |
|--------------------|------------------|----------|-----------------------|
| 8:1 MUX | CD74HC4051M96 | 3 | 0.08 |
| 3:1 MUX | TS5A3359DCUR | 1 | 3.6 |
| Hall-Effect Sensor | DRV5053VAQDBZR | 24 | N/A (negligible) |
| 0.1 μF Capacitor | C0603C104J4RACTU | 28 | N/A |

The maximum current draw is about 90 mA per module, although in practice we have measured a typical value of 60-70 mA per module. At VCC = 3.3V, each module consumes a maximum power of about 0.3W, although again we have found it to be about 0.2 - 0.25W per module.

² Maximum values determined from manufacturer's datasheet for each part.

Figures 6 and 7 show the final fabricated and assembled module board³.

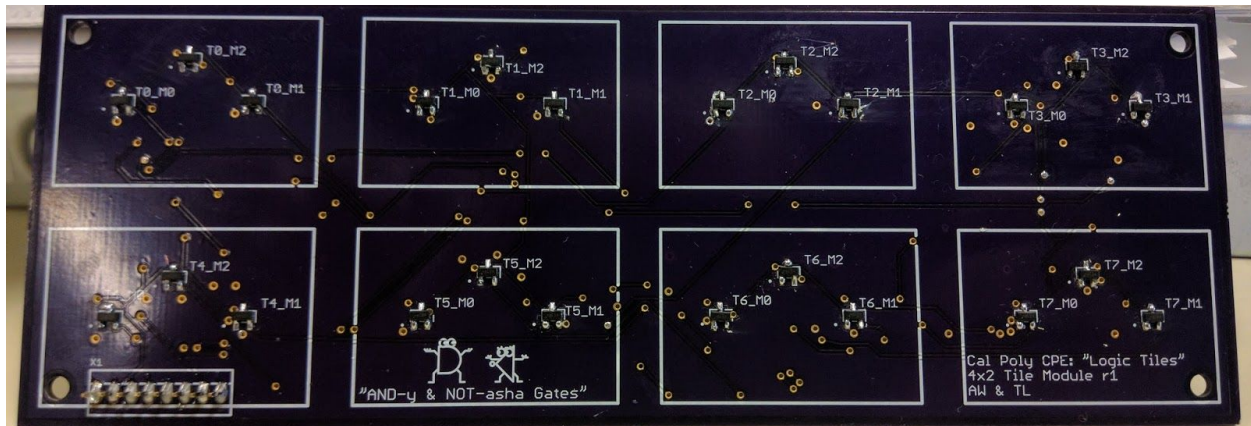


Figure 6: Physical Module PCB Front

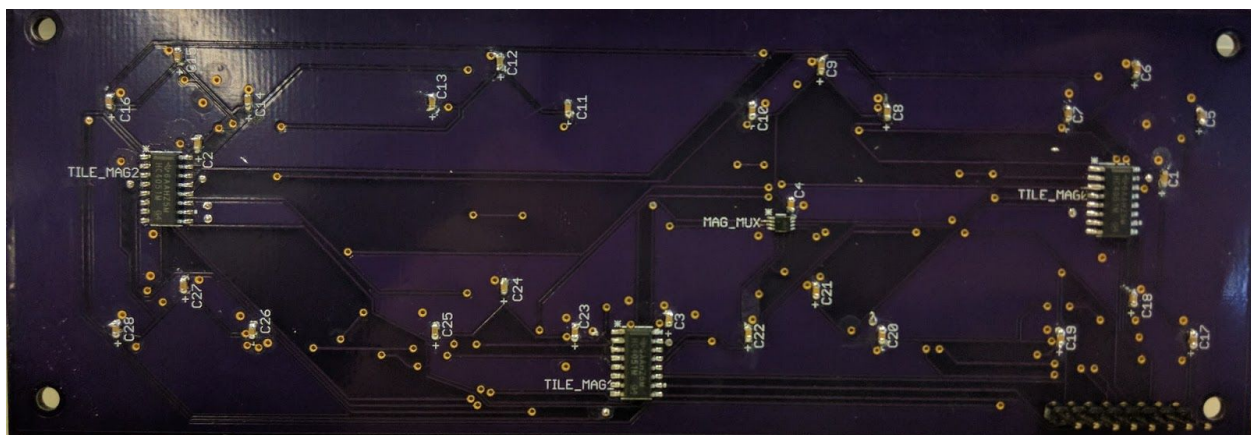


Figure 7: Physical Module PCB Back

We have additionally designed and created mounts for the modules that can be fit together to form complete tile boards of multiple modules. These mounts are available in the project files as Sketchup projects to be modified, or .stl files to be 3D printed.

³ AND-y and NOT-asha Gates are claimed as intellectual property by the authors, but may be freely reproduced non-commercially, or otherwise with express written permission.

Software Design

Data Structures

There are a few main data structures that store the data related to our system. One of these structures is the array of all the tile spaces on the board. This contains the current magnet value of each tile, the type of tile that the magnets resolves to, the orientation of the tile, and the circuit graph nodes connected to each side of the tile. These are stored as a `TileState` struct as seen below. These `TileStates` are stored as an array in which the tile number of the board is its index into the array.

```
typedef struct TileState {
    magcode mag0;
    magcode mag1;
    magcode mag2;

    tileType type;                // determined by the encoding of mag0-mag2
    int orientation;              // 1 is normal orientation, -1 is flipped (upside-down)

    struct Node *leftNode;        // not all will be used for each tile type
    struct Node *rightNode;
    struct Node *topNode;
    struct Node *bottomNode;
} TileState;
```

Another important data structure is the `Node`. These represent the connections between tiles and will be the primary method by which the logic circuit will be constructed and analyzed. These nodes contain information on which tiles they connect, whether it's been visited before (to avoid graph traversal loops), and the logical value at that node during traversal. There is also a `Node` pointer field for use in a free list of `Nodes`, as all possible `Nodes` are preallocated in order to avoid dynamic allocation.

```
typedef struct Node {
    /* Two tiles the node is bridging */
    TileState *tile1;
    TileState *tile2;

    uint8_t visited;              //for recursive graph traversal
    digiVal value;                //also for graph traversal
}
```

```

    struct Node *next;                //for use in the free list
} Node;

```

State Machine

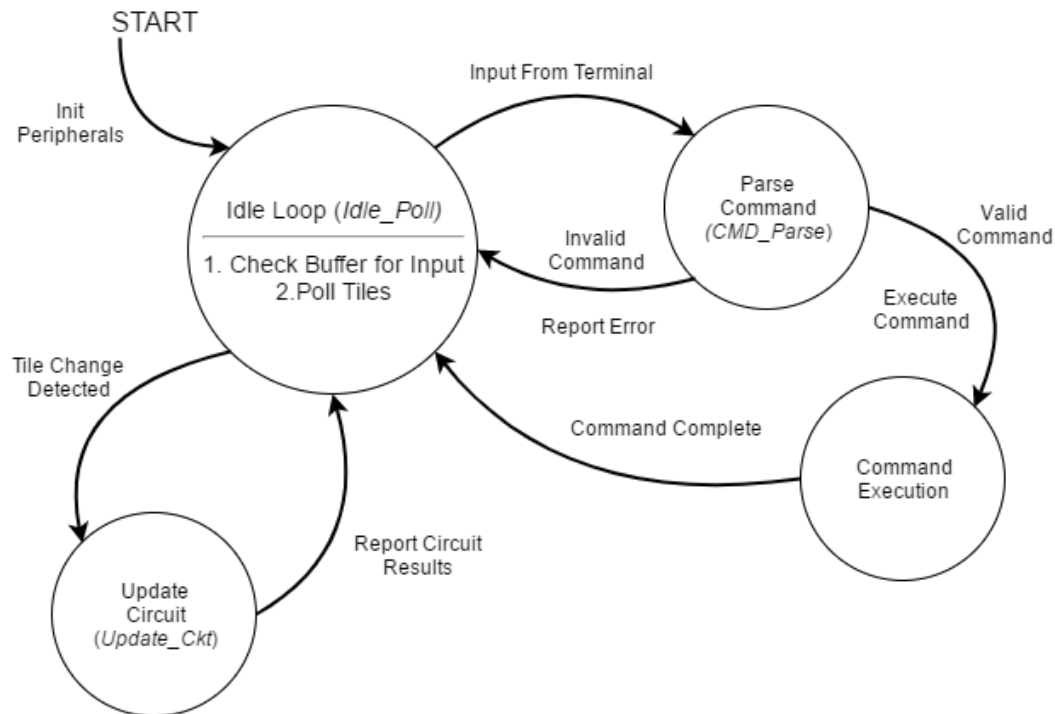


Figure 8: Software State Diagram

As diagramed above, the main control of our firmware is dictated by a finite state machine that's implemented in our main function. The state machine contains three main states: *Idle_Poll*, *CMD_Parse*, and *Update_Ckt*. The following describe the functions of all the states:

Idle_Poll

Idle_Poll is the default state that handles idle operation of our system and checks for changes in the current state of the board. The first thing it checks is the data input flag that is set if there is UART data available for reading. If there is data then *Idle_Poll* will return the state *CMD_Parse*, otherwise it will begin polling the tiles for changes. The act of polling the tiles involves reading the M2 magnet of each tile on the board and comparing it to the stored value. If it differs then the entire tile is read (all three magnets) and *Idle_Poll* will return the state *Update_Ckt*.

CMD_Parse

As the name suggests, *CMD_Parse* parses any commands that come in through the UART connection. If the string matches a known command, then that command is executed, otherwise it ignores it. This function will always return to *Idle_Poll*.

Update_Ckt

Update_Ckt is the state that handles the reevaluation of the digital circuit the user created. If a probe is present (one needs to be present to evaluate the circuit) it will begin the circuit traversal algorithm described below.

Circuit Traversal Algorithm

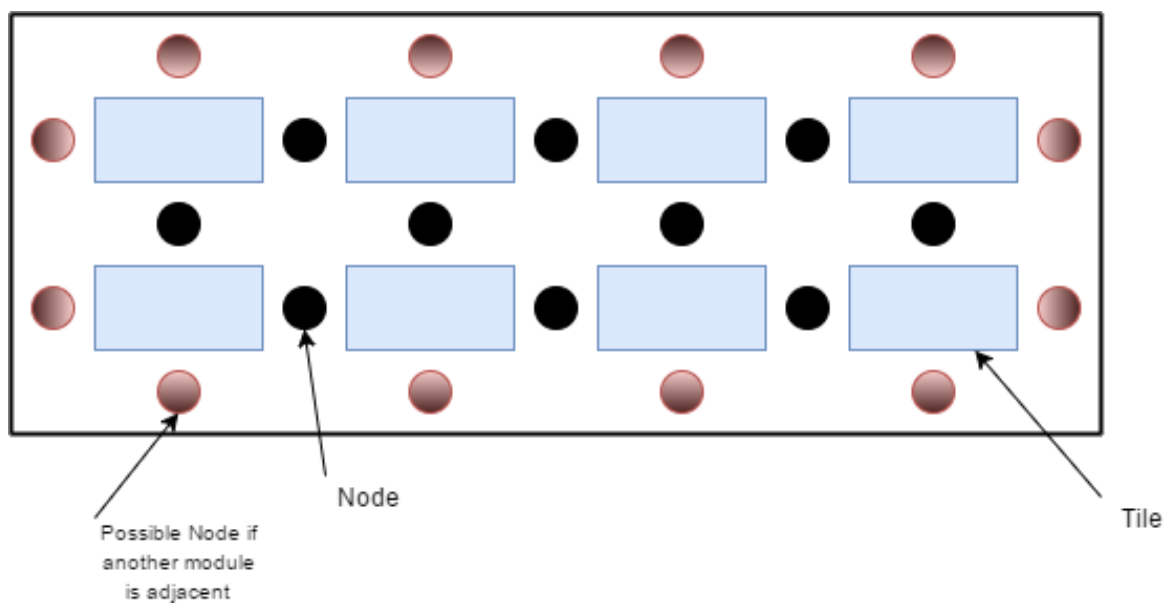


Figure 9: Graph Node Layout in Relation to Physical Tiles

The algorithm is a recursive graph traversal, with source tiles and dead ends serving as base cases. Each traversal starts with a call on whatever probe tile is currently being considered, as well as its left node (where the probe connects with other parts of the graph). Each call of the function locates which side of the parameter tile the parameter node belongs to. What next happens depends on the type of the tile and which side of the tile that node happens to be. In general, however, it will make further recursive calls on the other relevant nodes of the

parameter tile, and then perform an operation on the return values based what type of tile it is. The result of this operation is then returned. Below is high-level pseudocode of the algorithm:

```
digitalValue : {ONE, ZERO, INDETERMINATE}

digitalValue getNodeValue(Node n, Tile t) {
    digitalValue returnValue = INDETERMINATE;

    if (n != NULL && t != NULL) {
        /* determine the tile type of t */
        /* determine which side node of t is n */

        digitalValue temp1, temp2, temp3;
        Node sibling1, sibling2, sibling3;
        Tile neighbor1, neighbor2, neighbor3;

        if (n is not visited) {
            /* visit n */

            /* Note: not every node of t will be recursed like this. The type of the tile and
             * side of the node will determine which siblings/neighbors to make a call on */
            sibling1-3 = nodes of t that aren't n
            neighbor1-3 = the tile that each sibling node connects t with (NULL if none)

            temp1 = getNodeValue(sibling1, neighbor1);
            temp2 = getNodeValue(sibling2, neighbor2);
            temp3 = getNodeValue(sibling3, neighbor3);

            n->value = returnValue = /* Perform operation on temp1-3 based on type of t */
        }
        else
            returnValue = n->value;
    }

    Return returnValue;
}
```

The return values correspond to digital logic high (1), low (0), and indeterminate. Source tiles will not end up recursing, but immediately return their currently set value. Dead ends are the same, but with an indeterminate value. The operations performed by each tile are as follows:

1. Gates check that there is at least two (for AND, OR, XOR) or one (NOT) non-indeterminate values as operands. If there are not enough valid values, the operation returns indeterminate. Otherwise it performs the operation across all valid inputs and returns the result.

2. Wires check that there is one (and only one) value that is non-indeterminate. If there aren't any valid values, it returns indeterminate. If there is more than one valid value, it signifies bus contention, and returns indeterminate

Implementation / Results

Testing

In our system there are two main systems that need to be independently tested: the correct detection of tile types and the correct graph creation and traversal of the desired digital circuit.

Tile Detection

The testing procedure to determine the correct interpretation of tiles is relatively straightforward; we simply placed each encoded tile on the board and used our terminal interface to report what type of tile was detected at that position. If the tile was correctly typed, then we'd test to determine if our system could detect if the tile was flipped. If our system could not correctly detect the correct tile type, then we could have the raw ADC values output to the terminal. This would allow us to determine if it was the tile itself that was producing a bad reading of the magnet or if our software was incorrectly typing the tile based on correct read values. This also allowed us to build ADC read ranges for each of the given encoding values.

Circuit Creation

After being able to correctly detect a tile placement, we must make sure that it is integrated properly into the underlying graph data structure. This involves incremental testing that begins by creating the simplest logic gates possible: simple one gate circuits. Testing involves reading the connecting nodes of each tile to make sure the graph is well-formed, and then running the traversal algorithm from the probe in order to make sure it recovers the correct result.

To begin, we use a logical NOT with one input signal and a probe. After this test succeeds, we'd then try the other gates (AND, OR, XOR) with two inputs. For each gate we'd try all combinations of inputs, verifying that the gate's truth table can be implemented. When all gates work properly we can begin stringing multiple gates together and attempt to recreate the truth

tables for those circuits. The final test of our circuit's correctness is the building of a full adder. If that can be created, then we'll consider our system correct and successful.

Results

As of now, we can accurately detect and discern when a tile is placed on the board with reliable accuracy. We initially had numerous issues with the ranges we were using to detect certain magnetic field strengths. This is because our initial distance between the module sensors and tiles were too far, causing the magnetic field (and thus ADC read values) to be fairly weak, and the resulting ranges for each encoding value to overlap. Ensuring proper ranges of the encoding values is the most challenging and critical aspect in making this project work. We did not have the time to recreate a mount for the modules that would bring it close enough to the tiles, but we were able to mitigate overlaps for a successful demonstration in two ways:

1. Decreasing the depth of the X2 holes in order to move their ranges away from the U value.
2. Adding a second magnet onto the X1 magnets in order to distance their ranges from the X2 ranges.

It's important to note that these are temporary solutions done in order to produce a proof of concept that can be furthered with later iterations of this project. Decreasing the module sensor distance from the tiles is the only permanent (but very achievable) solution.

In addition, the graph creation and traversal algorithms all work as intended. The only hiccup in testing them was unreliable tile detection, as described above. It should be noted that we only ended up fabricating and connecting three modules, as opposed to our vision of six. This is because of the expense and time in fabricating additional boards. This project was not sponsored, and so we were constrained with standard project funding. That said, integration of three additional modules should only involve changing the board dimension defined values in the source code, as well making the appropriate physical connections.

Further Implementation / Next Steps

We believe that we have made our logic circuit board a solid foundation for further implementation to be made upon. For starters, we believe that a good feature to implement

would be to transcribe the circuit into HDL that can then be uploaded to an FPGA. Writing HDL and viewing it on an FPGA board is one of our main reasons for this project and having it in turn create HDL would have this project come full circle. Students could also use this computer generated code to corroborate their own code.

In addition, a large design component to be explored is an auxiliary board that serves to interconnect each module, multiplexing them together instead of needing to really on a different ADC pin per module. This could also be a production board for the MSP430 (or whatever chosen microcontroller) to run on, instead of TI's development boards. Forming reliable connections between the microcontroller and each module, including good grounding and Vcc, was inordinately difficult with breadboard and crimping, and this would address that issue as well.

Another important feature that would be good to implement would be sequential logic. As of now, our system only works with combinatorial logic. While this is fine, we believe that this project will more useful as an educational aid if it can implement sequential logic gates such as latches and flip-flops. These are typically more difficult topics for students and the visual representation provided by our project would be an effective aid.

Important alterations to our final design and system to be considered are as follows:

1. Fixes to the module mount in order to close the distance between sensors and tiles, as described in the preceding section. This includes appropriate depth changes to tile magnet holes in order to create the most distinct ranges.
2. Possible alternative sensors. Our sensors have a voltage range of 0-2V, regardless of Vcc. With the above fixes to tile and sensor distances, this should be plenty of range for the encoding values. However, if more encoding values or wider ranges are desired, using a hall-effect sensor that bases its ranges on 0-Vcc Volts would allow for it more easily. No changes should be needed to PCB layout either, as the footprint and package used is fairly common for these types of sensors.
3. Support structures on the underside of the tile faceplates in order to keep them from bowing inward.

Conclusion

In closing our involvement with this project, we have achieved the main goals that we first laid out. We have demonstrated how a system of tiles may be designed so that they are encoded with varying magnetic field strengths, and interpreted with a cluster of hall-effect sensors. Furthermore, we have shown that they may be encoded and interpreted in such a way to allow their orientation to be taken into account. The data structures and algorithm we implemented also successfully construct and interpret the digital circuit the user creates on the board. The terminal interface we've created provides a fairly intuitive method to assign source values and discern output values. We were even able to keep our system relatively cost effective; in the end our system cost around \$220 to produce.

There were several difficulties that presented themselves throughout the project, especially including the later stages. In any system like this, it is imperative to ensure that the ranges of magnetic field strength for each encoding value are all wide enough, and non-overlapping with each other. We struggled with this through the end, and an important point of focus for any continuation on this project would be the value ranges. We were also unable to fully assemble a 2 x 3 module board as we originally envisioned. Both time and minimal funding prevented us from doing so, but we hope that the groundwork laid out is sufficient enough to encourage future work to make it a reality.

In the end we consider this project to be a success. We've accomplished the goals we set out with and have created a system that we are proud of. There are plenty of additions to be made, so we hope that this is a project that another group of students would want to continue. Besides its implementation aimed at digital circuits, this system could also easily be adapted to work with any form of tiles, given the basic recognition techniques and graph structure that underlie it. We believe that this product has large potential and that it can be a useful aid to students learning digital design in the future.

Works Cited

- [1] T. McNerney, "From turtle to Tangible Programming Bricks: explorations in physical language design", *Personal And Ubiquitous Computing*, vol. 8, p. 5, 2004.
- [2] T. McNerney, "Tangible Programming Bricks: An approach to making programming accessible to everyone", Master of Science, Massachusetts Institute of Technology, 1999.
- [3] A. Smith, "Tangible Cubes as Programming Objects", ICAT'06, 2006.
- [4] A. Smith, "Using Magnets in Physical Blocks That Behave As Programming Objects", Meraka Institute.
- [5] Texas Instruments, "MSP430FR58xx, MSP43059xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide", MSP4305994 Datasheet, Oct. 2012 [Revised Jan. 2017].
- [6] Texas Instruments, "DRV5053 Analog-Bipolar Hall Effect Sensor", DRV5053 Datasheet, May 2014 [Revised Dec. 2015].
- [7] Texas Instruments, "Single-Channel 3:1 Multiplexer and Demultiplexer", TS5A3359 Datasheet, Oct 2005 [Revised Jan. 2016].
- [8] Texas Instruments, "Logic Analog Multiplexers and Demultiplexers", CDx4HC405x, CDx4HCT405x Datasheet, Nov. 1997 [Revised Feb. 2017].