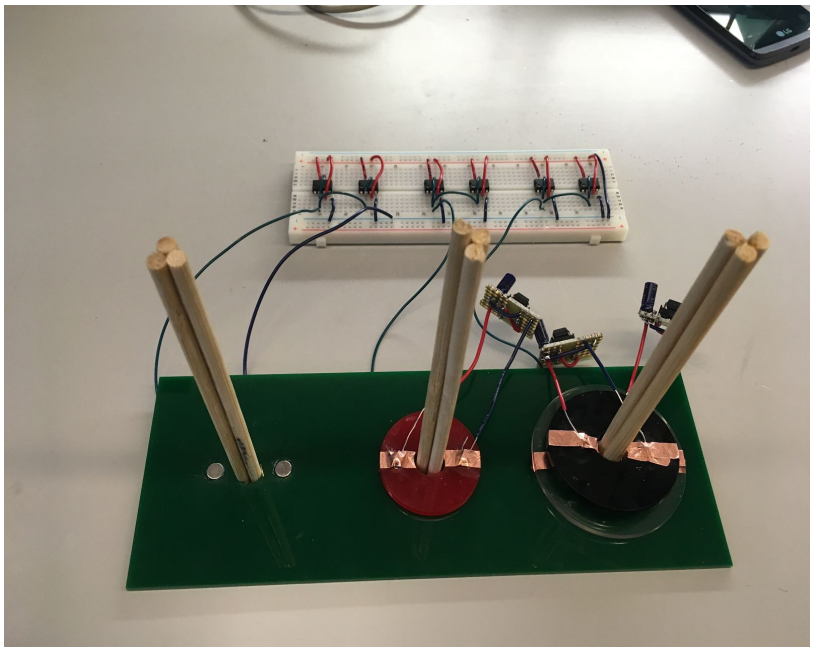
AtomWire Project

ESE-519: Real Time Embedded Systems

Siri Anil, Sana Kamboj, Sarvesh Patkar



Motivation

Conventional toy building block sets are well known and are generally considered to be an important part of a child's learning and development process, allowing children to use imagination and/or creativity to build and/or create a large number of configurations and/or structures. It would be desirable to create a set of blocks that can be used for construction in a conventional, physical way but can also directly interact with virtual platforms in real time by displaying a mirrored digital shadow of the block configuration. To build such applications, our project focusses on Atom-wire protocol which is built on top of 1-wire protocol. This serial protocol uses only ONE line for power and data.

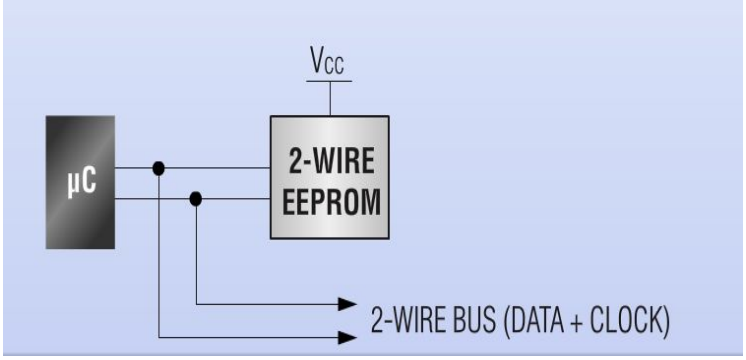


Fig: 2-wire communication between master and slave

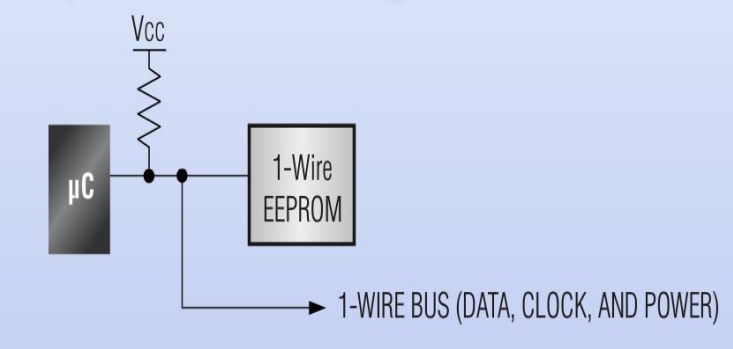


Fig: 1-wire communication between master and slave

Goal

To build sprinting, portable slave units using ATtiny85V for hassle-free and energy efficient communication with the master (Arduino). When the slaves are disconnected, it should receive power from the parasitic power supply (diode and capacitor cicuitt). When voltage returns, slaves wake up and signal their presence. When these slave units are coupled into the system, they should receive feedback concerning their new state and arrangement with respect to other slaves in the system. We demoed this energy efficient interactive platform with the use of two games (mini-chess and Tower of Hanoi).

Methodology

- Study the existing documentation and code base.
- Test the current ATtiny85V circuit on a breadboard. Find out the power requirements of the system such as time required to boot it, capacitor sizes enough to sprint, coordination of sleep/wake times amongst blocks.
- Study and test AtomWire protocol on a single MCU.
- Test AtomWire protocol on multiple slave MCUs on the same bus.
- Develop a simple game application on Unity.
- Scale the system to handle more slave units.
- Devise a mechanism to ensure connectivity of embedded devices while ensuring smooth physical gameplay.
- Develop the mechanical parts for the games.
- Integrate the game with the master MCU to receive inputs from the slaves and display on the screen.
- Design and develop the game on Unity and use the integration mechanism to get flawless non laggy gameplay.
- Perform debugging and stress testing.

Instructions to run the games

1. Program each of the slaves, with a different ROM code. Each slave gets a unique ROM code
2. Program the master (Arduino Uno) with the master code. (This code works as a polling code and polls through all the available slaves)
3. Connect the master (Arduino Uno) to the laptop with Unity game.
4. In Unity, change the port number in the script to that of the master's COM port. (This can be checked in the Arduino IDE).
5. Run the Unity game.
6. Now, arrange the slaves on the board.
7. Once they are arranged, the pieces in the game will arrange themselves to the positions shown.
8. Now, start playing and have fun!

Project Components

Software components:

- Unity Games (C# scripts)
- Arduino firmware code (C++ header files)

Electrical components:

- ATTiny85 slaves
- Arduino Master
- Capacitor-diode pair to keep the slaves up and running
- Op-amps (741)

Mechanical Components:

- Magnets
- Laser-cut acrylic slave base
- Laser-cut MDF game pieces
- Shot-glasses (:P)

1-Wire system

The project makes use of the 1-Wire protocol. The most notable aspect of 1-wire system is that unlike other systems, this uses only 1 bus for both power and communication. The communication happens only when the master controller pulls the line low. The slave consists of a capacitor which powers the slaves during the communication phase. Which means, the capacitor discharges when the line is low. When the line goes high, the capacitor charges again. The figure below shows the typical 1-wire system using the Dallas DS2413. As we can see, it has a capacitor built-in. Since we are using AVR ATtiny85V controller as slaves for our project, one of the challenges is to design the capacitor which can do the same function.

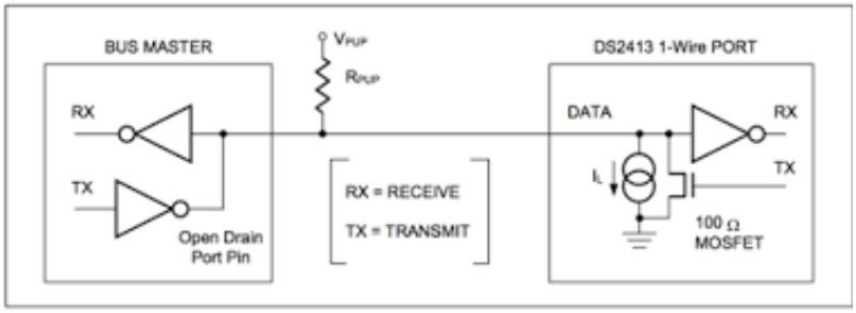


Figure 3: A high-level overview of the master and one slave. The DS2413 is a legacy part, and the circuit it encapsulates should be considered as part of an 1-Wire PCB.

Atom-wire Protocol:

Atom-wire protocol is built on top of 1-wire protocol. This serial protocol uses only ONE line for power and data. Atom wire protocol updated for AVR devices with additional functions like GPIO Read/Write etc. The updated code has been updated on github. The 1-wire commands used in ATomwire protocol have been explained below. This protocol is

- Energy efficient
- economical
- Hassle-free (few wires)

In our applications, we have used Arduino Uno as master and AVR ATtiny85 as slaves. Using this protocol, we designed portable slaves - disconnecting the slave puts it in a defined reset state. When voltage returns, slaves wake up and signal their presence.

1-Wire commands:

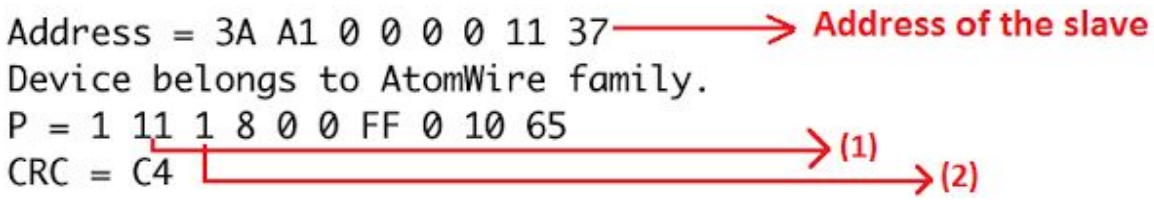
These commands detail how the master device specifies which slave device or devices it will communicate with. These are the standard 1-wire commands. They are summarized below.

- READ ROM [0x33] - This command requests the serial device of a slave. If multiple slaves are on the network, data collisions will occur and the data received will not be valid.
- MATCH ROM [0x55] - The master must immediately follow this command with an AtomWire ID that precisely specifies one slave on the network. If multiple slaves have the same ID, data collisions will occur. All slaves that do not match this ID will ignore AtomWire communications until a Reset pulse is detected
- SEARCH ROM [0xF0] - Initializes all nodes on the AtomWire network that an ID search is about to begin. The standard search algorithm is given in the 1-Wire specification, though it may be modified (see project description).
- SKIP ROM [0xCC] - Allows the master to skip specifying the an AtomWire ID if there is only one slave on the line. Data collisions occur if multiple slaves are connected
- RESUME [0x69] - Selects the last AtomWire device that was successfully selected via a MATCH ROM or SKIP ROM command.

These commands were tested on the 1-wire AVR slave and we got the expected results.
To extend the work done so far, we also added few commands which were not in the 1-wire specification.

Atomwire Block IDs

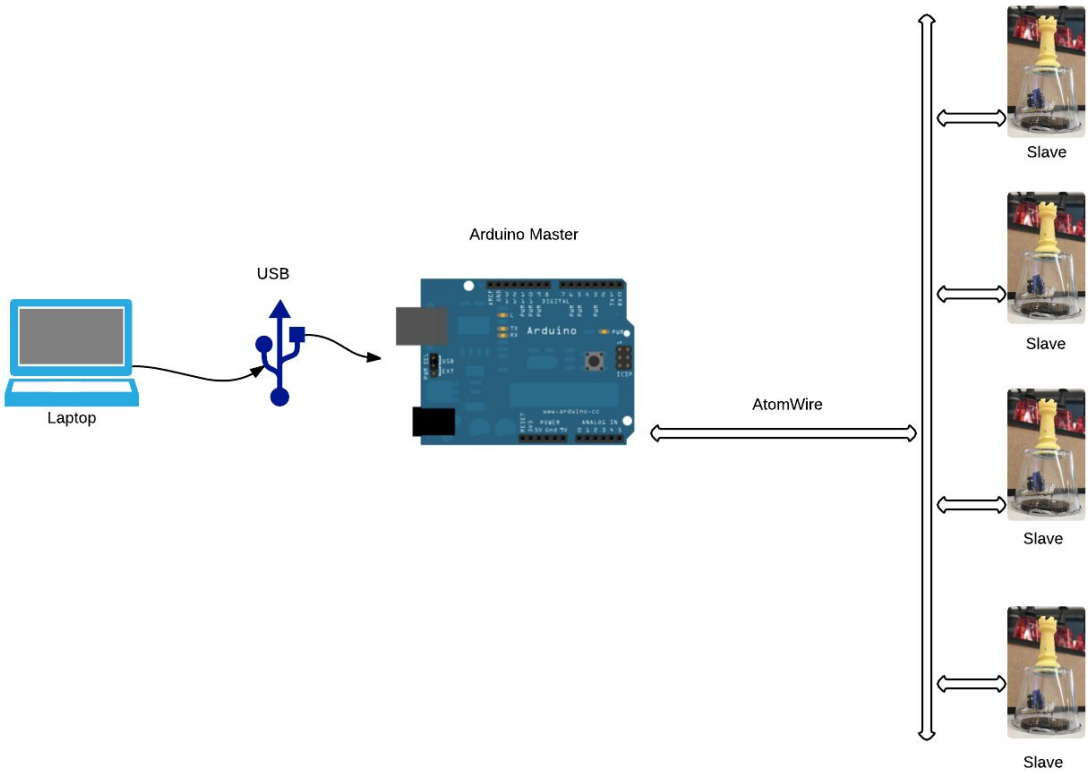
Each slave has a unique AtomWire ID. The ID encodes both specific details about the type of slave and an identifier that is unique to that particular slave. The master is able to tell the slave to which block the slave belongs to and also the slave's ID in that block. This is depicted as follows:



Here, (1) signifies the type of the network. For example, the above screenshot tells that the slave is in a 1x1 network. Other types of networks can be 2x2, 2x1 etc.

(2) signifies the ID of that block in the network. Since the above block is in 1x1 network, there can be only 1 slave in the network whose ID is 1. If it was a 2x2 network, then there would be 4 slaves and each slave will take the ID 1,2,3,4.
The main idea of this is to assign different IDs to the slaves by the master to the slave so that it can communicate with them as necessary.

Architecture

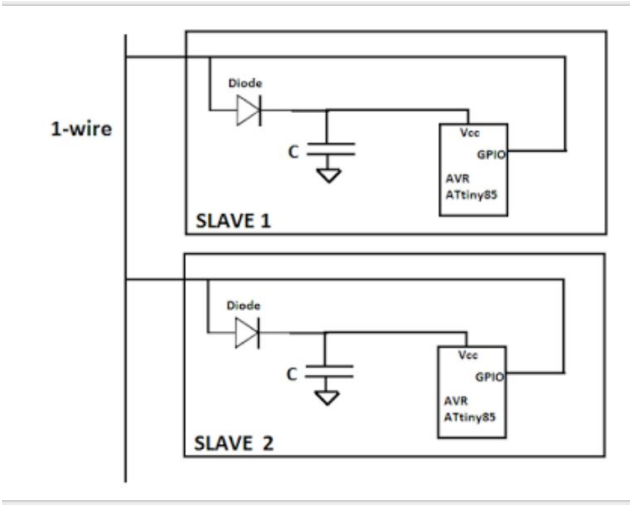


- The laptop (with the game) communicates with the Arduino Master through serial port.
- The Arduino master evaluates the responses it receives from slaves on its bus and sends appropriate commands to other slaves. It also replies back to the master with content
- Each slave receives 1-wire commands to perform some function through the atomwire line

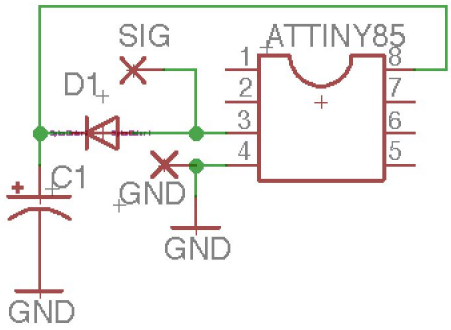
Hardware

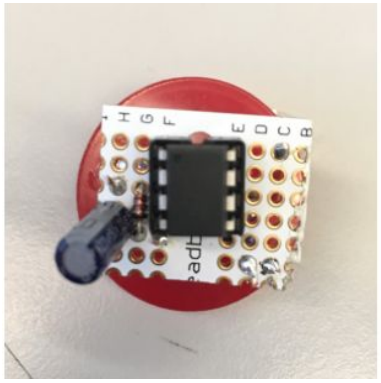
Slaves: Each slave has its own parasitic supply - a capacitor and a diode. The GPIO pin is connected directly to the one-wire bus. VCC is connected to the bus through the parasitic supply.
Slave - AVR ATTINY 85

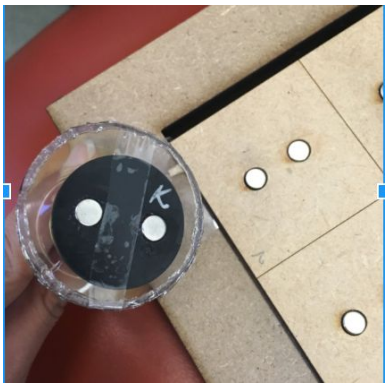
Diode - IN4148
Capacitor - 22uF
GPIO pin used - pin 4 of AVR ATTINY 85

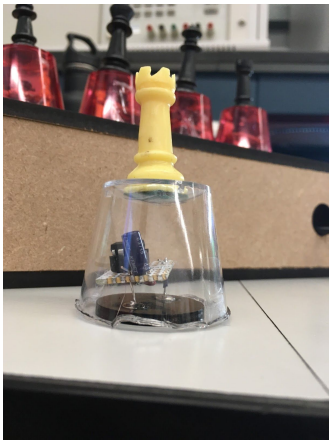


Portable slaves - General purpose boards are used to build the slave component. The one-wire line and gnd of the slaves are attached to two magnets of opposite polarity. These magnets are are designed to align with the one-wire line and gnd reference on the master side board.





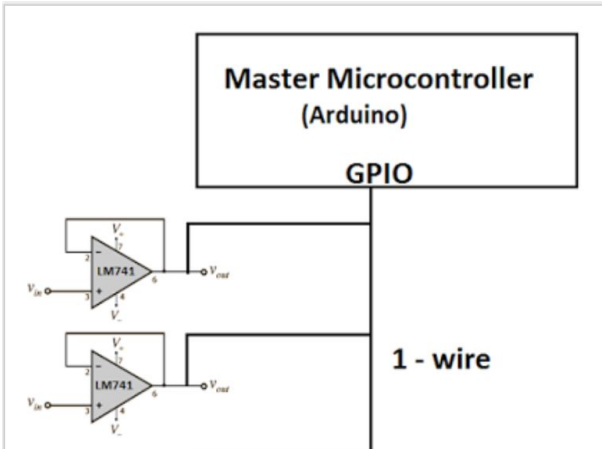




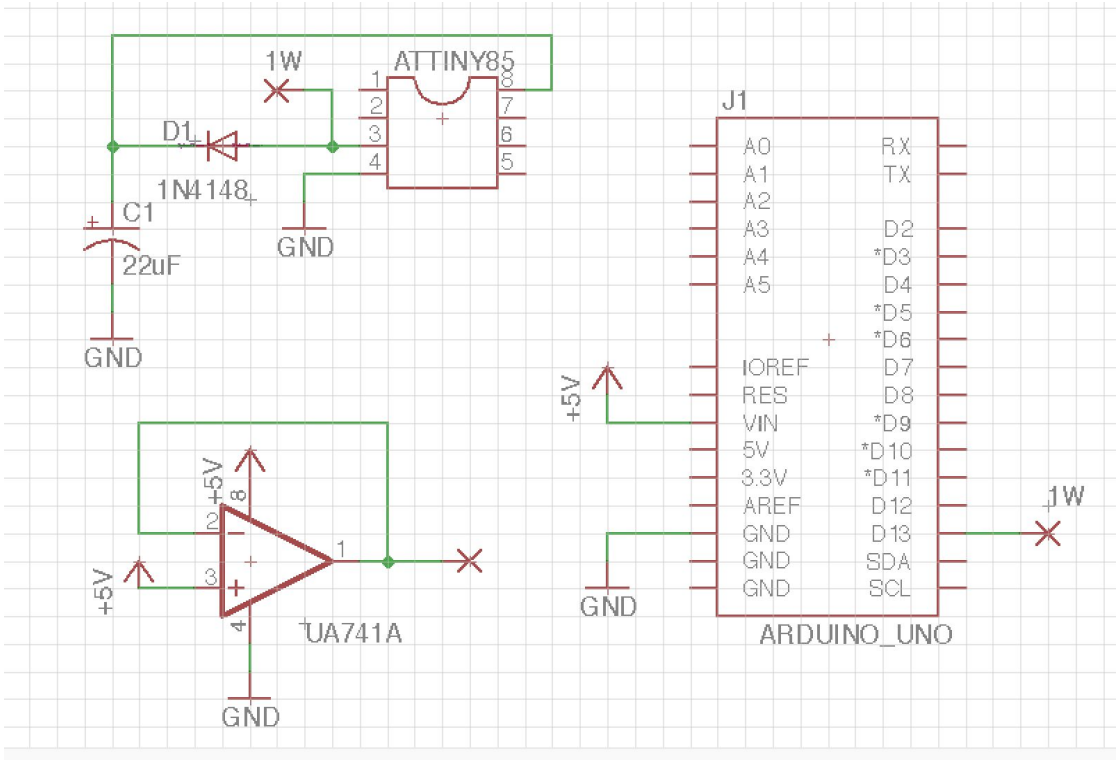
Master: One GPIO pin of the master is connected to the one wire bus. In one-wire protocol, a resistor is used to pull up the one-wire line. For our Atom-wire protocol applications AVR microcontrollers are used as slaves, so this pull-up resistor acts a series resistance to the inner impedance of the slave. This reduces the volatge at the Vcc of the slave below the operting volatge of the slave. Hence, voltage follolwers are used instead both for pull-up and isolation purposes.

Master - Arduino UNO, GPIO pin - pin 13

Op-amp - LM 741, V+ : 5V of arduino, V- : ground

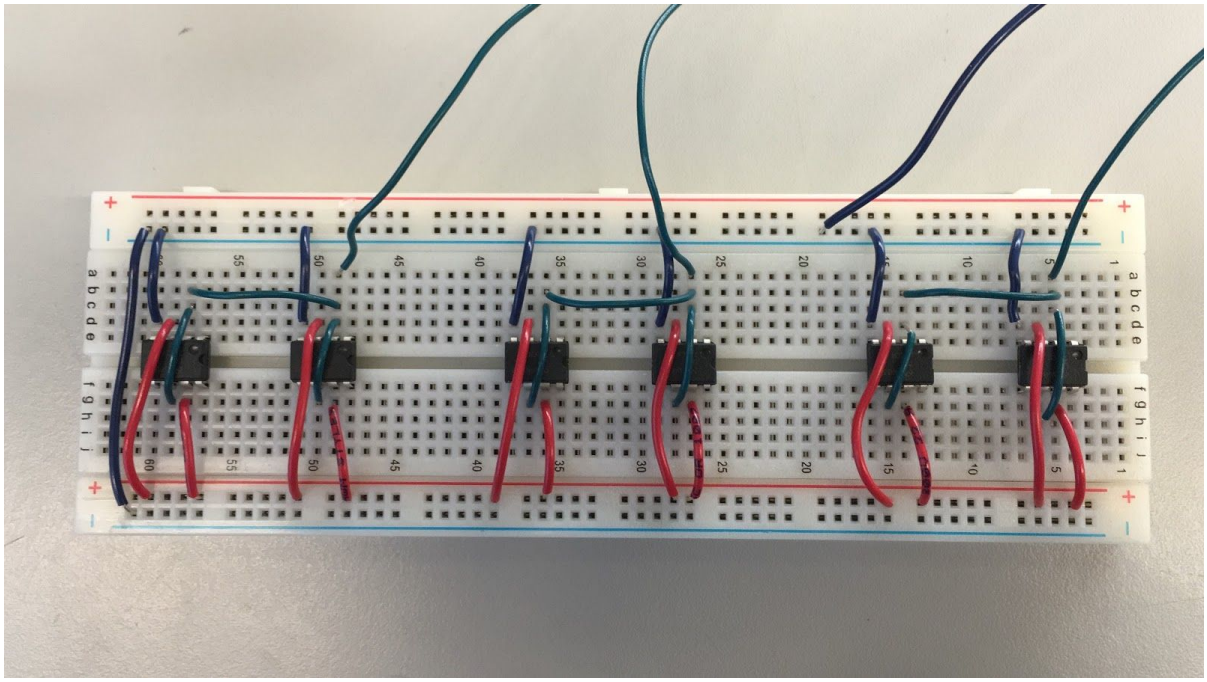


Master-slave circuit



Multiple slaves

This setup allows upto 4 slaves on a single one-wire line, with two volatge follwers connected in a similar fashion. This system can be scaled up by using additional GPIO line of the master as a additional one-wire line. Hence, there is no resistriction on the number of slaves.



Software

Unity

The front end design for the game was written in Unity. Unity is a cross platform game engine developed by Unity Technologies.

The steps that we took to program the unity game was as follows:

We installed the Unity game, the version that is available to download on the Official unity website. Following this, based on what we already knew about unity and what we knew from our requirements, we came up with a simple yet effective game visual that would get the display the power of the One Wire protocol to the viewer. We came up with 2 designs - 1. Tower of Hanoi 2. Chess

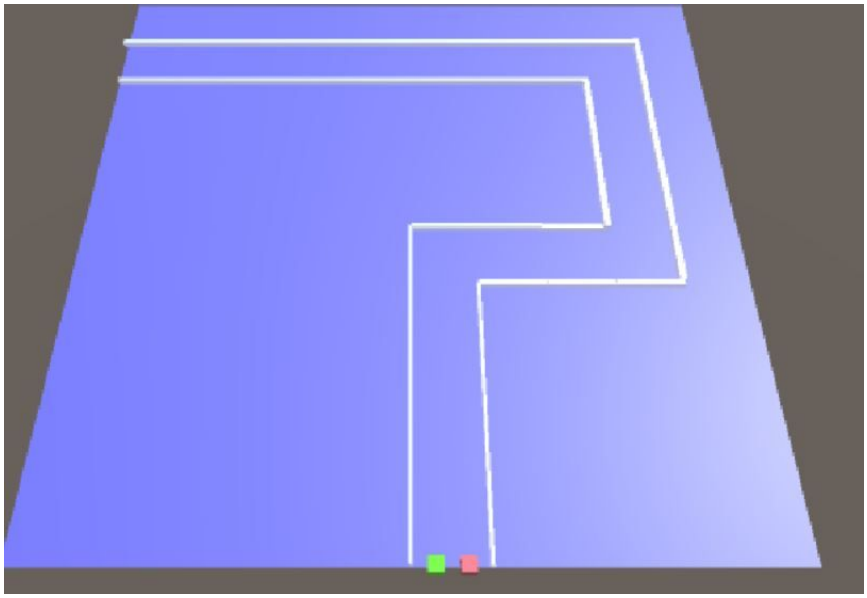
For the chess game, the first step in building the game was to decide the layout. Since we decided on 4x4 chess, we made the chess board using 16 cubes that were adjusted in the right proportion to get the layout of the board. This was followed by giving each of the blocks a shade that would resemble an actual chess board. This was done using the material context in unity. This gave the entire structure of the board.

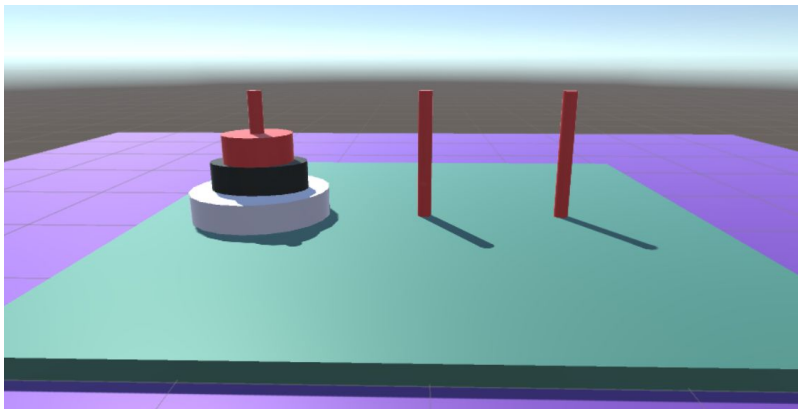
The second step was to arrange the pieces. Because of the time constraint, we could not design the chess pieces ourselves. So we imported them from the unity asset store. We had many designs to choose from, but we chose the one that matched our chessboard well.

We then arranged our chess pieces in the right format on the board and the entire structure was ready. Our next step was to get the pawns to move according to the input from the master arduino. To do this we used the serial communication of arduino to send messages about its current state to Unity. Unity would take this as the input on its serial input and use it to move the pawn. Each of the packet that was being sent by arduino was sent in a particular format and then decoded on the unity side. We used Csharp scripts for the front end coding.

The tower of hanoi had a similar interface as the chess game. The thing that put them apart with respect to the front end design was the kind of objects we used and what we did with those objects. For the towers we used cylinders, that represented our towers in hardware. To do the pieces, we had cylinders with appropriate dimensions. Based on the input sent by the arduino we would use it and display the objects appropriately.

Our biggest takeaway with respect to the front end was the opportunity to work with a new tool such as unity and explore the various things that we can achieve through it.





Arduino software, libraries

The existing code base had Arduino libraries for the AtomWire protocol. These libraries We incorporated a counter to denote state in the scratchpad memory. Every time the user disconnected the slave the counter got reset. So, to maintain state throughout the session we tested EEPROM storage on the ATTiny85. Data stored on the EEPROM can be stored and reread once the slaves wake up and pushed to the scratchpad.

Mechanical Parts

Magnets

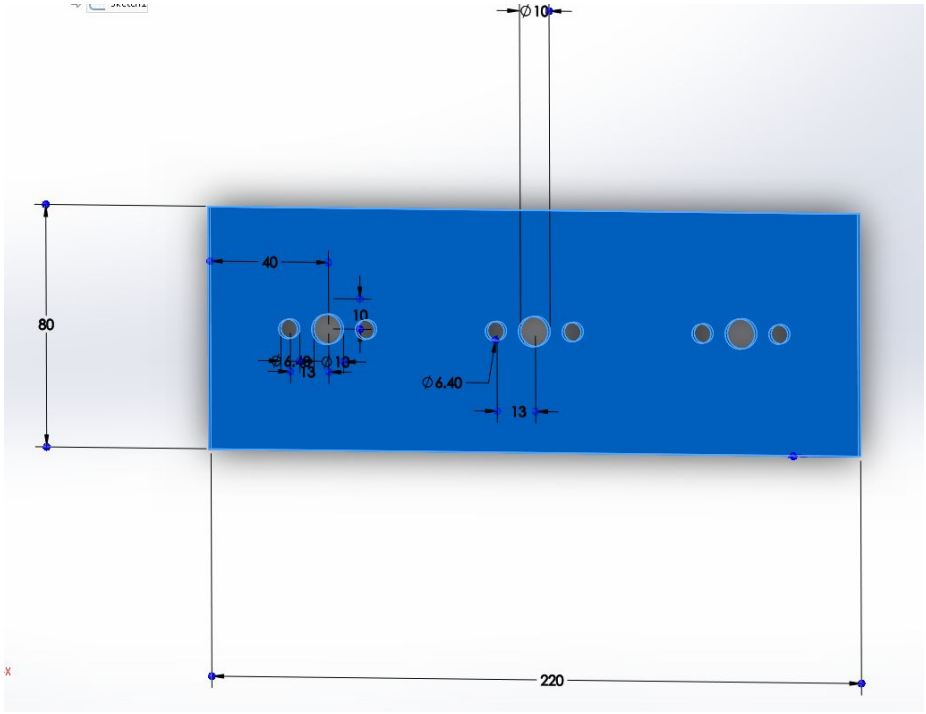
Tiny 6mmx3mm magnets were used to ensure electrical conductivity and orient the pieces correctly when being placed on the board. Out of the 2 magnets, one was the power/data line of the atomwire master. The other was the ground line of the master.

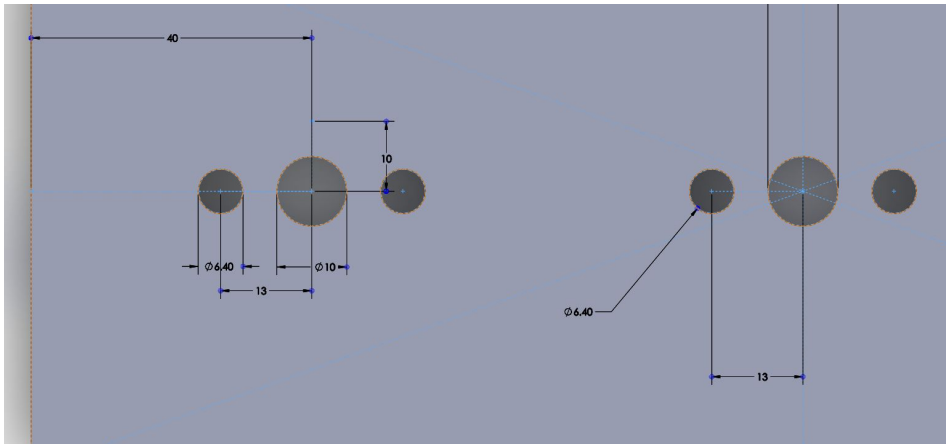
These cheap magnets proved to be really critical to the working of the system as the entire system relied on the slaves being connected to the master correctly.

Laser cutting

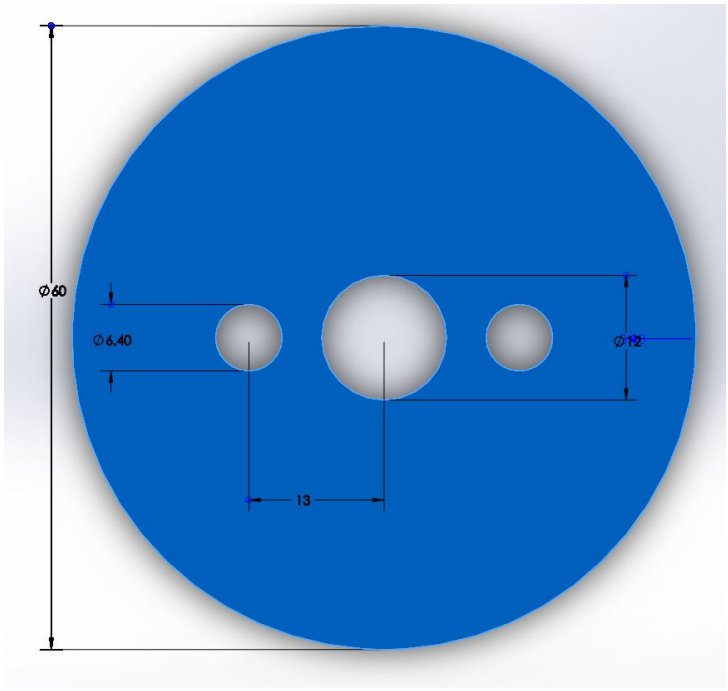
The following parts with dimensions were laser cut.

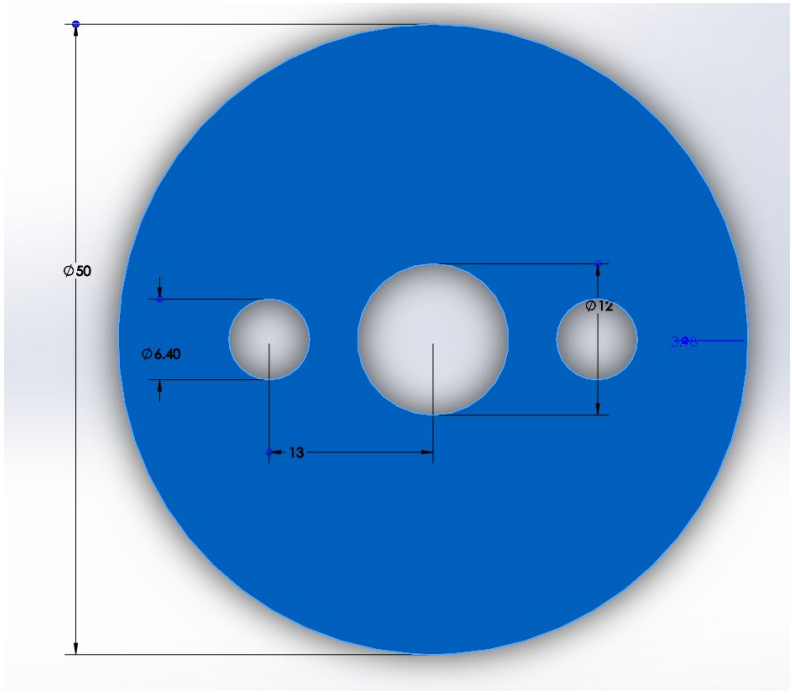
1. Base of Tower of Hanoi

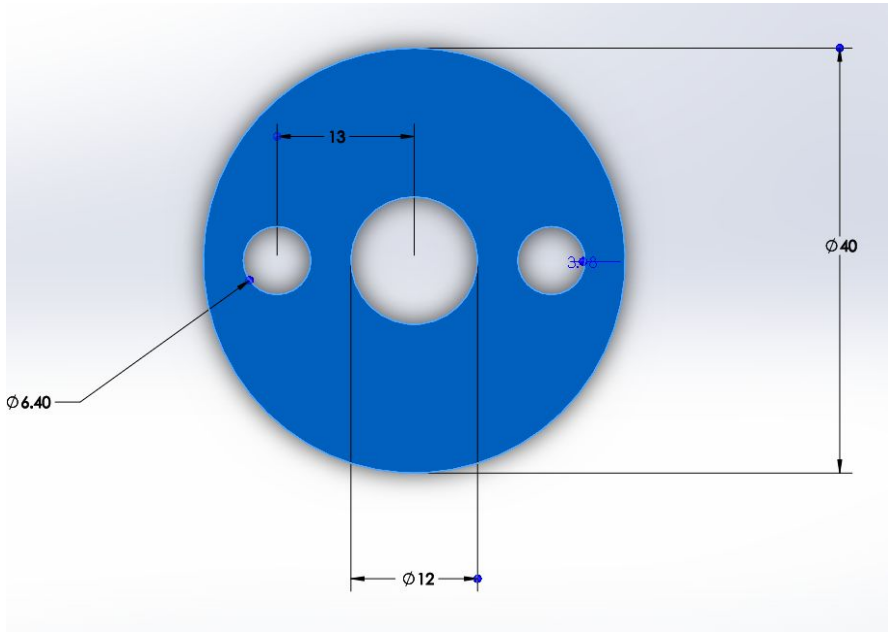




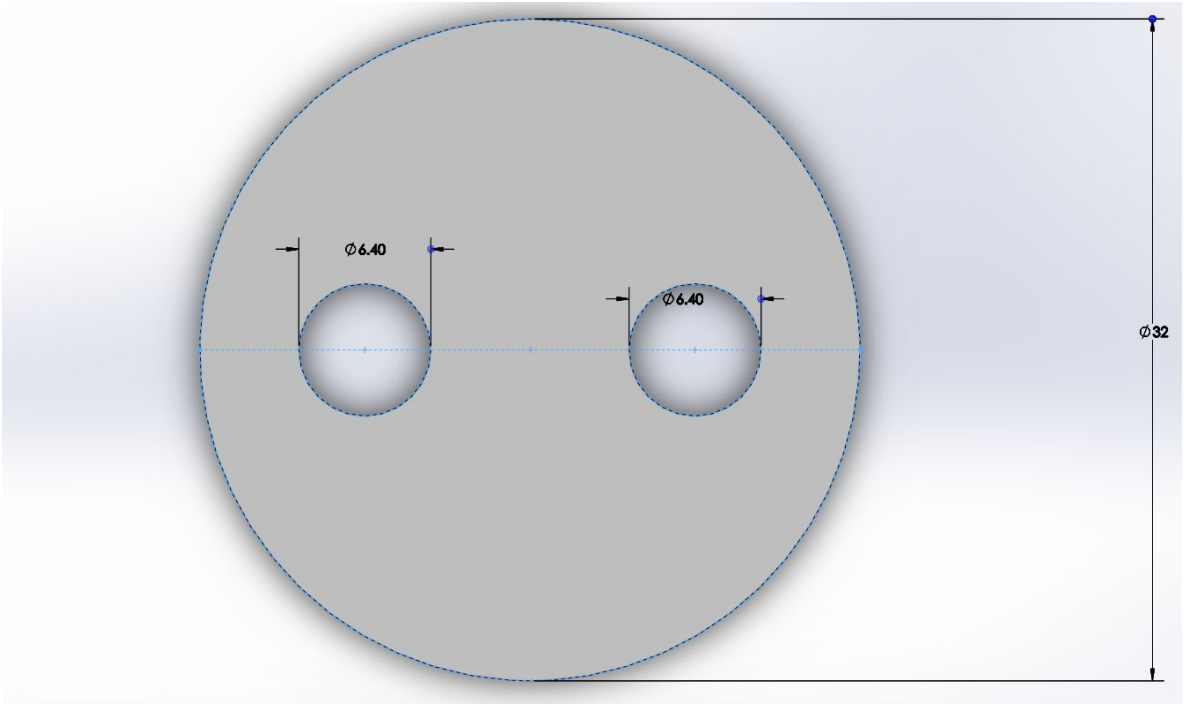
2. 3 discs


Ø60
Ø6.40
Ø12
13

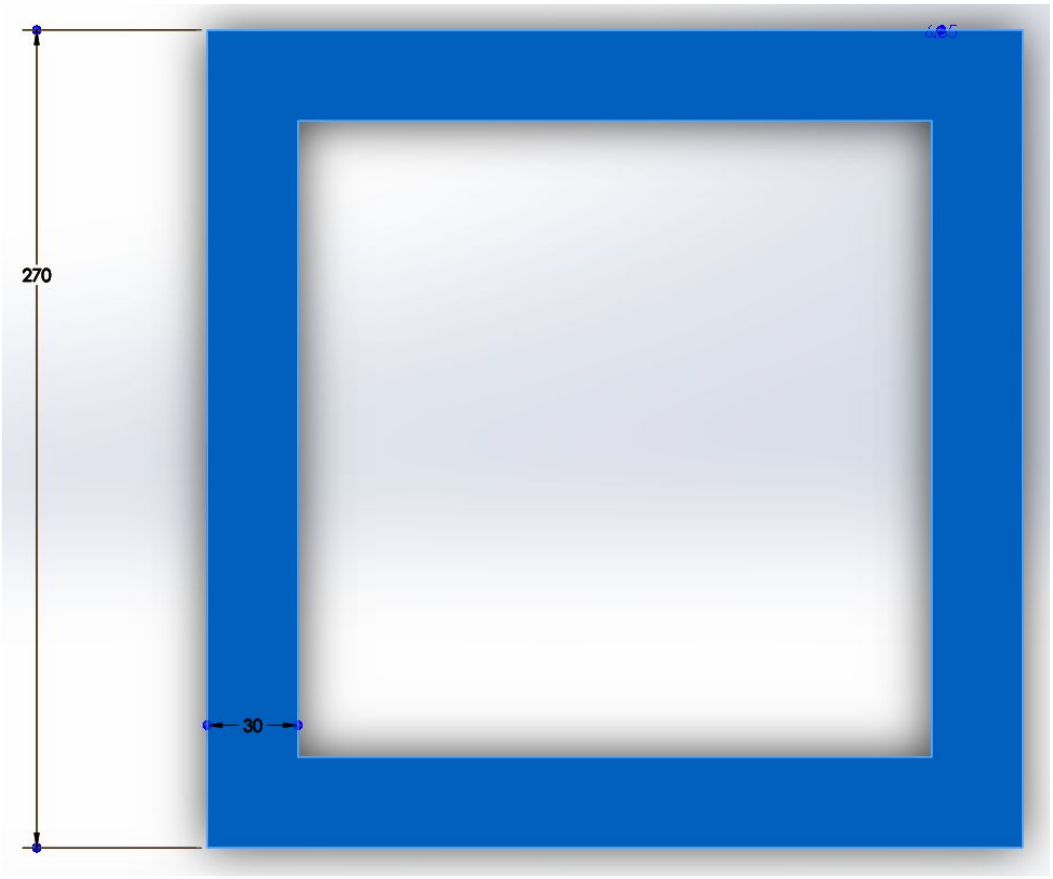

Ø50
Ø12
Ø6.40
13

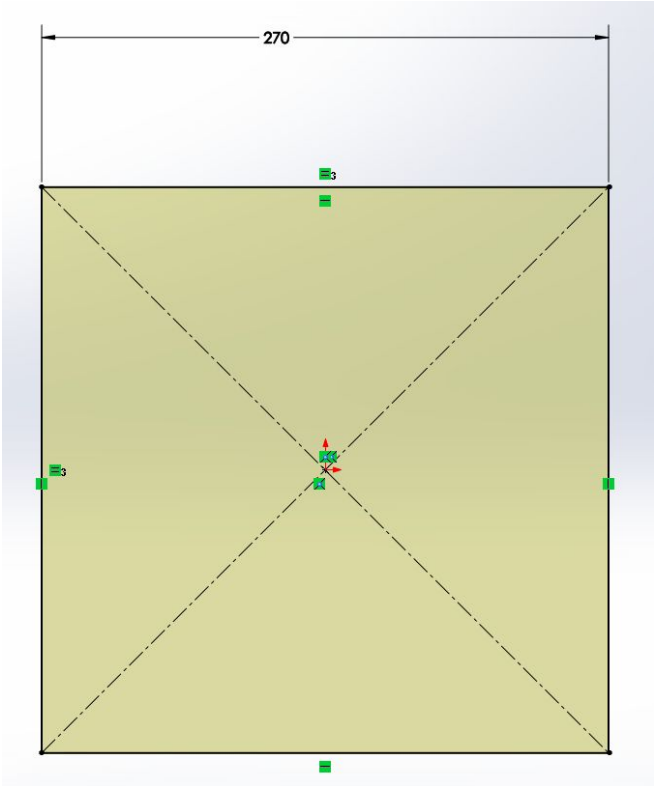


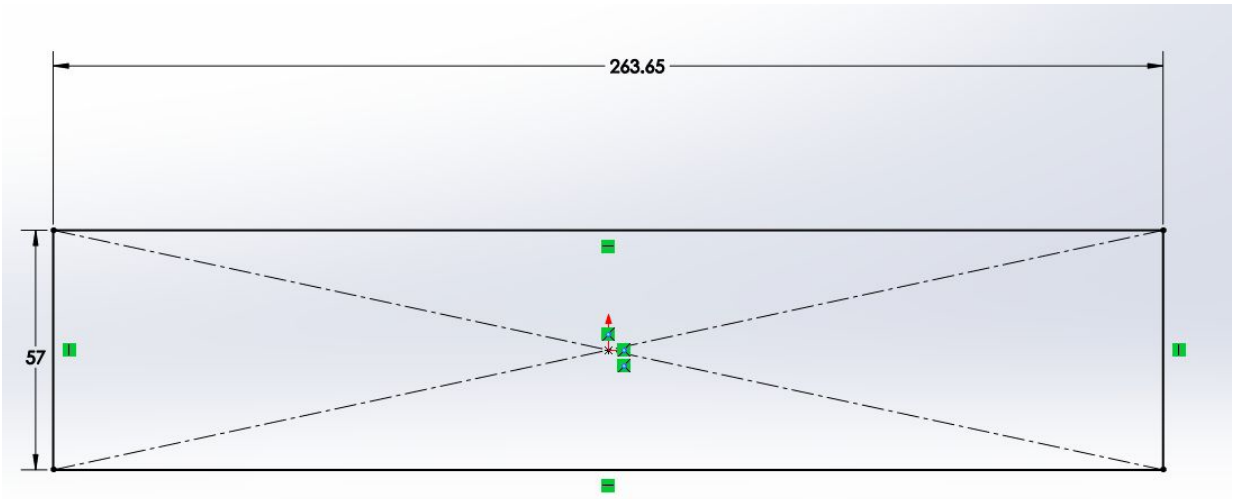
3. Chess slave pieces

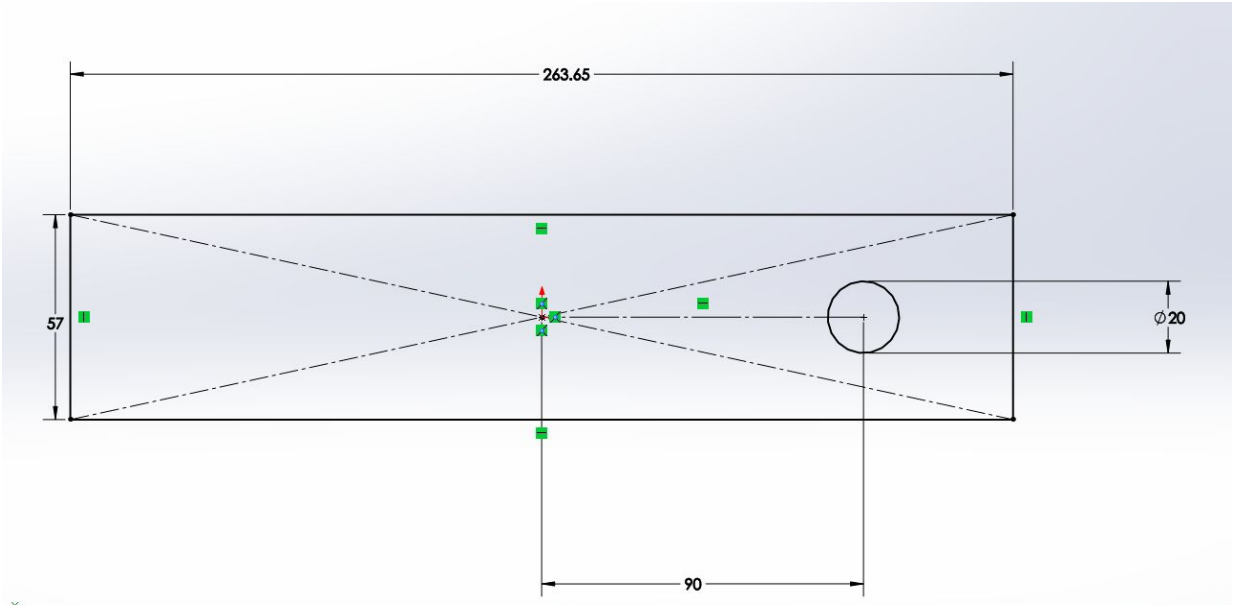


4.Chess box parts


270
30


270





Testing

Testing and debugging was done at each step to evaluate whether a certain component would be more useful than the other for the project. Integration testing was done by checking connectivity between the slave and the master. The oscilloscopes were used to check whether the Atomwire protocol was being followed correctly. Debugging was done by using pre processor directives to keep a track of whether a certain section of code was getting executed. This was especially helpful when the Unity engine refused to listen to Serial data and it was essential to fine-tune what was being sent over the USB Serial port.

Conclusion

Two games were designed to showcase the scalability and the power of using 1-wire communication. Having intelligent blocks as toys in both the virtual as well as physical space is a good way to develop child's skills in this age of computers in both the virtual and physical space. The games were designed end-to-end with only the protocol specifications as a reference. Thus, following further steps can be taken independently to develop the protocol further.

1. Build an even more scalable application of Atomwire that syncs with cloud and can allow multiplayer gameplay feature
2. Improve the state maintaining method of AtomWire by using an even better way than EEPROM
3. Add more functionality to AtomWire. Make it run as a task on a RTOS so that the microcontroller slave using it can be made more energy efficient.