Board Buddies: A Physical Board Game with Wireless Communication Capabilities

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Andrew Kremp

Spring, 2023

Technical Project Team Members

Ahmed Hussain

Daichi Monma

Emily Parnell

Richard Zhou

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harry C. Powell Jr., Department of Electrical and Computer Engineering

Statement of work:

Daichi Monma:

My responsibility was mechanical design. It involved making the housing, tiles, tile sensors, and integrating those with the main printed circuit board (PCB). We decided to make the sensors for each tile to be buttons, so I came up with the idea of using double pole buttons to index a tile and reduce the number of inputs to the microcontroller unit (MCU). I did schematic and layout of the button PCBs that each make up a quadrant of a board. I made an order with a PCB manufacturer. To keep costs down, I chose to use an alternative board house. Once arrived, I populated the button PCBs with their parts and wired them together to make prototype that allowed others to test their software. From around the button PCBs, I designed the housing. Taking this design, I used Autodesk Inventor to model every part of the device. Using its computer-animated design (CAD) environment, I planned mounting holes for the PCBs to bolt onto 3D-printed frames and the frames to a wooden enclosure. I led the effort on 3D-printing components by first fixing available printers and then taking an STL file of a part and slicing it using a program called Ultimaker Cura to operate the printers. It required calibrating and experimenting with printing settings to manufacture. I recycled wood from past projects and used a table saw and band saw to make the wooden enclosure.

Andrew Kremp:

During this project, I was primarily responsible for all things involving communication. This includes wireless communication as well as universal asynchronous receiver-transmitter (UART) communication. In terms of wireless communication, for our final design we decided to employ the use of two Raspberry Pi Pico W's, one dedicated to each game board, as well as a central Linksys Wi-Fi router. I was tasked with configuring the Picos to send data back and forth, wirelessly, through the router. In order to do so, I first configured the router to assign each Pico a static internet protocol (IP) address, so we knew which Pico was which. I then wrote code in the MicroPython programming language using the Thonny integrated development environment (IDE) to connect both Picos to the router's network and allow them to communicate back and forth via a user datagram protocol (UDP) connection. This way, the two game boards could send game state information back and forth without any wired connection, which was one of the major purposes for pursuing this project.

Since we decided to use MSP430FR2355 microcontrollers to program the game logic and interface with our external gameplay components (pushbuttons and light-emitting diodes (LEDs), UART communication was needed to send information between the MSPs and Picos. Thus, I was tasked with writing the functions in the C programming language using Code Composer Studio (CCS), to utilize the MSP's UART transmit and receive buffers to send data to/receive data from a Pico when we needed to within our gameplay finite-state machine (FSM). Similarly, I was tasked with writing functions on each Pico that allowed us to receive data from/send data to each MSP when desired. As an example, when one player makes a move, that move needs to show up on the other person's board. My code takes the game state data from one

MSP, delivers it to the corresponding Pico, sends the data to the other Pico wirelessly, who sends it to that Pico's corresponding MSP to update the other player's board. I worked with Ahmed and Emily to determine the contents/setup of the UART message.

I also worked with Richard to complete most of the testing regarding our gameplay FSM. This involved setting up a liquid-crystal display (LCD) for each board, which are controlled by the Picos to display custom messages depending on the current state of the FSM (this LCD is also used for the actual gameplay beyond testing). Testing also involved playing through whole games with Richard to catch any problems with the FSM code, find any hardware issues with the printed circuit boards (PCBs), and ensure everything works as expected. We tested a series of gameplay edge cases (e.g., moves that the code may accidentally consider valid that really shouldn't be) as well as potential wireless/UART communication vulnerabilities (e.g., handling inappropriately dropped bits in messages).

Ahmed Hussain:

My primary role was acting as the embedded software architect for the project. In the beginning of the project, I laid out the general structure of the software to be written for the MSP430 chip as well as how it would interface with the Wi-Fi transceiver. This included making a basic FSM to model functions of the board and ensuring the microcontroller satisfied memory requirements. The basic FSM was then further developed by Emily to include every possible state in the board system.

I worked with the physical MSP430 PCB to ensure the 64 buttons could be read without 64 inputs to the microcontroller but rather 16. To do so, the 8 rows and 8 columns were multiplexed together by activating two 3.3V lines whenever the corresponding button index was pressed. Reading this input also required mapping the software to the correct pins such that a row activation and column activation could be used together to locate the tile pressed. This also worked against accidental multi-presses of buttons. I worked with Richard on this part of the project.

I also worked on the wireless communication aspect, particularly the message which was to be communicated between boards. I worked with Andrew to develop a communication message structure which could encode the Othello game state, game requests, turn indicators, and any errors which may have occurred on the other board.

Richard Zhou:

My primary role was designing and testing the electrical hardware of the project. I designed the input protection, power supply, and microcontroller circuitry of the PCB. Since our group used an MSP chip rather than a Launchpad, I researched the Spy-By-Wire method of debugging the MSP430FR2335 we were using and ensured that we would be able to debug and flash code to the PCBs. I was also responsible for doing the layout and revisions of the board. Once the boards arrived, I was responsible for assembling the PCB and testing the components of the PCBs. During the hardware testing phase, our team encountered some hardware and software issues that I helped deal with by reworking the PCBs.

I also helped with figuring out how to interface with the Wi-Fi module (in this case the Raspberry Pi) through UART. This was then implemented onto the MCU PCB. I also worked with Andrew to test both the wireless code as well as various parts of the FSM. This involved testing various scenarios that can occur in Othello as well as playing through full games to ensure that the win and lose states were reached correctly.

Emily Parnell:

My two primary responsibilities were the light-emitting diodes (LEDs) for our display and designing and writing code for the game FSM. I picked out the LEDs and the connectors/components relevant to the LED subsystem and made the schematic sub-sheet for this part of the board. Richard laid the subsystem with the rest of the PCB and soldered the parts to the board, and Richard and Daichi both soldered strips of the LEDs for them to lie in our housing correctly. I also wrote the code that would control the LEDs. The LEDs take a SPI clock and data to determine how many LEDs light up and how bright and what color each should be. I wrote functions to initialize the pins for LED clock and data and to send the signals which make the LEDs light up in the desired way (right colors/brightness at the right locations). After Richard soldered the board, we tested the LEDs together to debug hardware and software issues. When the full housing was assembled, I adjusted the colors and brightness of the LEDs to be more suitable for our final product.

I designed the FSM for our game, determining which states were needed, the output at each state, and the criteria for transitioning to another state, which was comprised of user inputs and game logic. There were three elements of game logic. The first was determining if an attempted move was valid and updating the game state if it was. The second was determining if a given player had any valid moves, as a player relinquishes their turn if they have no moves, and the game ends only when both players have no moves. And the third was determining the winner at the end of the game. I wrote functions to make these calculations, as well as test functions to verify them. I then wrote the code for our FSM, which maintained the current state as well as having functions for computing the next state and sending the output. I worked with Andrew when adding the pieces concerning UART (when the FSM relies on sending and receiving messages) to make sure this was integrated correctly. Our inputs and outputs were abstracted well enough that I could incorporate them into the FSM without needing to know the details of the input and output code.

Table of Contents

Contents

Table of Figures

Abstract

Board Buddies allows two people to play a physical board game while in different parts of the world. The system is comprised of two game boards (one for each player) that can communicate wirelessly through a central server. When one player makes a move on their board, that move is sent to the other board, and both boards are reconfigured appropriately. During the game, players interact with an 8x8 tile display interlaid with pushbuttons and LEDs. When a player wants to make a move, they will push down a tile on their board, and the color of the appropriate tiles will change in accordance with the gameplay instructions. The system will moderate the players' choices and not allow an illegal move or allow one player to go twice in a row. It will do so in the backend code then communicated externally through an LCD. Board Buddies is an internet of things (IoT) product [1], pairing software with hardware to push the boundaries of how a board game can be played.

Background

Our primary goal in pursuing this project is to create a unique spin on the traditional board game entertainment system. After the last few years, in which the COVID-19 pandemic forced almost all person-to-person communication to be digital, many are growing tired of purely online entertainment [2], particularly those that are less technologically inclined. However, experiencing this period provided many with a new perspective on the value of communicating with loved ones, no matter how far away they are. As such, we wanted to create a system that would allow people in different locations—long-distance couples, friends at different colleges, grandparents and grandchildren, etc.—to enjoy the fun that comes with a physical board game, even if they can't be in the same place to play it.

We also believe our system will open a new sector in an otherwise stagnant board game industry: a physical game board with modern digital and wireless capabilities.

There are many websites and apps which allow people to play board games with remote players from their desktop or phone, such as [3] and [4]. These sites also allow someone to play against a computer. However, both options require the players to use a computer or phone to play; alternatively, we aim to create a physical interface which would make the game feel more immersive.

There have also been precursors that allow users to play two-player games on non-computer/phone devices wirelessly. The Nintendo game link cable [5] was created to allow friends to connect their Game Boys and play a game together. Later, Nintendo created Game Boy Micro Wireless Adapter [6] which allowed two devices to connect without a physical cable between the two devices. Still, this kind of game connection between physical interfaces can only communicate over a short-range distance, which is much more limited than what we endeavor.

There are many reasons people may not wish to interact with a screen: they might want to avoid screen time in order to not impair their sleep [7]; they might want to avoid eyestrain as a result of screen use [8]; or they simply might long for a tangible board out of nostalgia (this is particularly

true of elderly demographics). But if someone wishes to play with a friend or significant other that lives far away, a screen-based game is the only option that exists. Our system will be the first to alleviate this issue.

This project will require us to employ our knowledge of several topics we have learned from taking courses in the University of Virginia (UVA) Electrical and Computer Engineering curriculum: embedded system design, computer networks/wireless communications, hardware design, power supplies, and signal processing.

Embedded system design will require using a microcontroller to control much of the gameplay (interpreting pushbuttons and lighting up appropriate tiles), as well as handling data from the board and sending it off to be processed by/receiving it from our server. Networks and wireless communications knowledge will be vital to transferring game state information from one board to the other. Hardware design skills pertain to the design of a PCB [9] to connect and control electrical elements of the design. In terms of power supplies, the game board will be powered via a wall outlet, and, as such, appropriately dispersing the power from the outlet will be needed to ensure the different board components work properly. Finally, signal processing knowledge will be needed to appropriately set up the touch sensors users will interact with when touching tiles on the board.

Courses group members have taken that pertain to these topics include Embedded Computing and Robotics Series (ECE 3501 and 3502)/Introduction to Embedded Computer Systems (ECE 3430), Computer Networks (ECE 4457) the Fundamentals of Electrical and Computer Engineering series (ECE 2630/2660/3750), and Wireless Communications (ECE 4784).

Ahmed, Andrew, and Emily are computer engineers and thus have an abundance of experience with embedded systems and computer networks. They have worked frequently with microcontrollers, like the one we used to control the state of the game, when designing projects for their courses and organizations (including the Solar Car Team [10], Gizmologists [11], and Cavalier Autonomous Racing organizations [12]). Ahmed and Andrew have also worked with networks and data processing and handled parts of the project that dealt with transmitting/receiving data. Ahmed has previously worked as a Firmware Engineering Intern at Tesla Motors [13], where he gained most of his embedded systems knowledge. Andrew currently runs the Gizmologists organization and works with UVA's Cavalier Autonomous Racing team, learning about networks and working heavily with embedded systems. Emily has designed radio frequency (RF) and quantum systems as a research intern at the California Institute of Technology [14].

Richard and Daichi are electrical engineers and thus have an abundance of experience with hardware design, power supplies, and signal processing. Both are also members of the Solar Car organization with Ahmed, which has given them extensive experience designing circuits in software called KiCAD [15]. This will help ensure our board elements are connected efficiently and safely. Daichi has worked as an Engineering Technician at M.C. Dean and Bluefinn Innovations [16], frequently working with hardware design and other practical electrical topics.

Richard has picked up useful mixed-signals electronics design skills as an Analog IC Intern at Jacobs Engineering [17].

Physical Constraints

Design Constraints

Unfortunately, due to some of the shortcomings in terms of part availability, manufacturing limitations, etc. there were some constraints we had to adjust for when finalizing the design.

For example, most parts took at least a week to come in after we ordered them. However, due to the relatively short time period we had to work on this project, we could not afford to wait a week for the parts to come in. Thus, we made sure to design our systems in parallel and try to make as many things testable without hardware components as possible. This involved testing the game logic, LED logic, and pushbutton logic all purely in software before we received the components.

Since we used an MSP430FR2355 [18] microcontroller for this project, which is provided by Texas Instruments (TI) [19], we at one point planned to use a TI specific Wi-Fi module (CC3200) [20] purpose built for interfacing with MSP microcontrollers. However, these modules were out of stock until January, so we had to pivot in another direction, purchasing Raspberry Pi Pico W devices [21] instead.

Lastly, we modified the design of our button inputs due to a limited number of input pins on our MSP microcontroller. With our initial logic, we would have required 64 inputs for 64 buttons per board. However, the MSP430FR2355 does not have 64 inputs available to use. Thus, we decided to try and multiplex the buttons such that each button was recognizable by a (row, column) index. Doing so reduced the number of inputs from 64 down to 16, which enabled us to connect the buttons directly to our MSP's input pins as desired.

Cost Constraints

There were a few aspects of our project that required us to be more cost conscious than other teams. The foremost of said aspects is while most other project groups that made board game systems only made one physical board (because they were programming a bot to play against one player), our system required two physical boards (because our system involves two human players). Thus, we had to get at least two of everything we ordered.

One of the major areas where our project saw more costs than most was PCB costs. In total, our final project required 10 PCBs to work (four tile PCBs and one full system PCB per board), not to mention the prototype PCBs. Thus, PCB costs alone probably would've put our project over budget if we had used the recommended Advanced Circuits option to produce them. Instead, we used a cheaper alternative located in China called JLC PCB [22] which saved us about $100.

We made sure to use recycled materials whenever possible throughout the project's development. As an example, most of our game logic testing was done on a recycled MSP430F5529 launchpad [23]. The powers chord we used to power the boards were taken from

two projects from the previous semester, and the board housings were made from wood also taken from a previous project.

A rough breakdown of the expenses is seen in Figure 1. There was about $100 spent on manufacturing the primary boards, $200 spent on parts for the primary boards, $30 spent on the pushbutton boards, $50 spent on the LEDs and LED connectors [24], and $70 spent on the pushbuttons and pushbutton board components. The remaining materials used in the project were scavenged. Without using recycled parts, the true cost of the project would be a bit over the $500 budget.

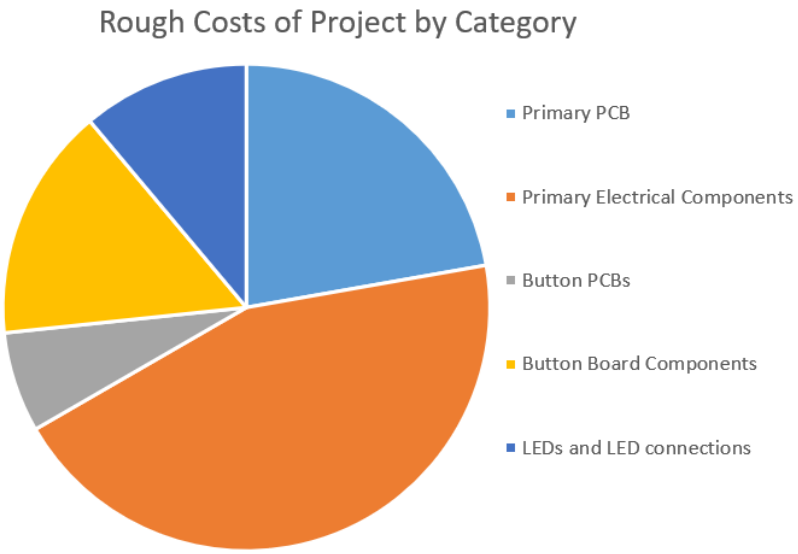


Figure 1 Rough Cost Breakdown

Tools Employed

For this project we used a series of software development and hardware design tools. On the software development side, Code Composer Studio (CCS) [25] was used for anything pertaining to the MSP430FR2355 microcontroller. Specifically, code pertaining to reading the pushbuttons within our tiles, our gameplay FSM, sending signals to turn the tile LEDs on or off, and UART communication between the MSP430FR2355 and Raspberry Pi Pico W was written in CCS. Code using CCS was done in the C programming language. All team members had experience with CCS through UVA's Introduction to Computer Embedded Systems (ECE 3430), but it took some time to reacclimate to the software.

We also employed the use of Thonny [26], an integrated development environment (IDE) that runs MicroPython. Thonny was used for anything pertaining to the Picos. MicroPython is the preferred language to use when setting up these devices. Andrew, the one who programmed the Picos, did not have experience with MicroPython before, but he did have extensive experience with regular Python programming, which made the transition seamless.

On the hardware design side, PCB design was done using a software called KiCad [27]. PCB designers Richard and Daichi used this software to create schematics for our PCBs as well as layout the PCB to send off to a manufacturer. Two manufacturers were used to manufacture the

PCBs: Advanced Circuits and JLCPCB. Advanced Circuits was used for the more complex PCBs that housed the microcontrollers and power supply circuitry (as they were a little more expensive but more reliable), and JLCPCB was used for the less-complex button PCBs (because their PCBs were around $100 cheaper than what they would have cost through Advanced Circuits).

Finally, for the design of the physical game boards, Daichi, the team member responsible for making these boards, used Inventor [28] to CAD a grill that created 64 individual tiles on each board. He also used a series of power tools, including a table saw, hand drills, and sanders, to make the wood housings for the boards.

Societal Impact Constraints

Environmental Impact

The environmental impact of the Board Buddies game boards is minimal, but nonzero. The main contributors to our environmental footprint are 3D-printed acrylonitrile butadiene styrene (ABS) [29] and polylactic acid (PLA) [30] and indirect stress to the supply chain. ABS and PLA are both Type 7 thermoplastics, which most recycling centers do not accept them, and they cannot be placed into recycling bins. This is a dead-end environmental impact, as they must be disposed of as opposed to being recycled. On the other hand, the supply chain experienced indirect stress from component and PCB orders, the environmental impact of which originating from the transportation and building of these goods. While the 3D-printed plastic cannot be considered environmentally friendly, many of our other miscellaneous parts were reused from other projects; this includes the wooden chassis and nearly all jumper wires in screw terminals. This is a positive environmental impact.

Sustainability

In order to promote sustainability for large-scale distribution of this product, a pamphlet informing the consumer of ways in which they can properly dispose of the board elements would be included to provide recommendations for recycling plants which can handle electrical and printed materials.

Health and Safety

This project has minimal impact on the health of its users, as it will be acted upon and will not exert any force or produce anything ingestible. The greatest safety concern we have is ensuring the electrical and mechanical components aren’t dangerous to the user in the sense of everyday use. Safety is the primary purpose of input protection and other circuit safety features we will implement, such as a power ON/OFF switch near the power cable. Before finishing the board, it was ensured to adhere to the appropriate safety standards set by the industry (see “Standards” section).

Ethical, Social, and Economic Concerns

The ethical considerations needed for this project are relatively minimal. As these boards were designed for consumer entertainment, we don’t expect any ethical issues beyond making sure that the game system is economically feasible. It is highly unlikely that a user could find much

use for the Board Buddies system beyond a game. One issue which could arise is the interception of network packets whenever the boards attempt to communicate back and forth. Deciphering these packets could lead to either leakage of personal information such as IP addresses or the possibilities of tampering with received messages.

External Considerations

External Standards

National Institute for Occupational Safety and Health (NIOSH): 3D Printing Safety

We used a 3D printed design for the body of the game boards, alongside a wood chassis. We did not use any heavy-duty industrial-grade printers, but it was still important to note safety while using even small 3D printers. The NIOSH and the Center for Disease Control (CDC) recommend the following 5 safety steps when using 3D printers: 1) characterization of potential hazards, 2) work activities, 3) describing engineering controls, 4) managing administrative controls, and 5) wearing personal protective equipment (PPE). This safety standard is important in the context of mass production, especially when many printers might be manufacturing simultaneously in the same enclosed space. [31][32]

National Electrical Manufacturers Association (NEMA): Mechanical Enclosure

The NEMA safety standard plays a “vital part in the design, production, and distribution of products destined for both national and international commerce”. Specifically, for mechanical casings and electrical enclosures, NEMA enforces a strict rating system against intrusion, dust, ice, oil, accidental bumps, and liquids. We need to meet NEMA’s Type 2 standard, which protects against incidental contact, limited amounts of falling dirt, and very light amounts of liquid water. Our product meets this standard, as it can withstand dust, intrusion, and accidental bumps; there are no open electrical leads, meaning minor amounts of liquid also do not affect performance. [33][34]

Institute of Electrical and Electronics Engineers (IEEE): Wireless Connectivity

We met the IEEE 802.11a/b/g/n standard (specifically 802.11n) to ensure we created a product which could be more widely accepted and used, as the standard will encourage interoperability between multiple manufacturers. Note that the 802.11 standard promotes low-cost wireless connectivity for internet of things (IoT) devices such as this product. This standard is also built-in to the Raspberry Pi Pico we used for wireless connectivity. [35]

International Electrotechnical Commission (IEC): Electrotechnical Safety

The IEC 60950-1 standard promotes electrical safety. We met this specific standard, as our product needed to meet the requirements of being mains-powered or battery-powered information technology equipment. [36]

Intellectual Property Issues

The Board Buddies system is a set of devices that uses Wi-Fi to allow users to play the game Othello remotely, while still maintaining the physical aspect of playing a board game. One patent

similar to Board Buddies is the “Remote Live Table Gaming Terminals and Systems” [37]. This patent is for a game table used in casinos which allows users to play remotely with users from other casinos. Like Board Buddies, this system operates on a network, allowing players to interact with users that are in a different location. This patent also includes the ability to accept a request to play in a game during a tournament. However, the major difference between these two systems is the game being played. In the case of Board Buddies, the game that is played is Othello, which is not found in casinos.

The patent for “Game Systems and Methods for Remote Card Games Using Physical Playing Cards” [38] describes a system where players can play a card game remotely while still using physical playing cards. This is done through card sensors that detect the card being played as well and send the information about the card. The system has many games states which are determined by the cards being played, like the Board Buddies system, in which the state of the game that each board sees depends on the move a player makes. Similar to Board Buddies, this patent uses wireless communications to communicate between players. While this patent shares similarities to the Board Buddies project, the main difference lies in the game being played. Board buddies plays Othello, which does not use cards and has completely different game logic than any card game that can be played on the system described in the patent.

The patent for “Gaming device having board and converting chip game” [39] is a game like Othello but slightly modified. In the game Othello, which is used in our project, the game does not finish until every tile on the 8x8 grid is occupied. In the game described in this patent, both players are assigned several moves. In the case of this patent, there is only one human player, since the player plays against the gaming device. When the number of moves is exhausted, the players chips are counted. The game in this patent shares similarities with the Board Buddies system in that both games include chips and outflanking the opponent. However, the main selling point of the Board Buddies system is that a user plays against a human opponent instead of a machine, which is different from this patent. This patent includes a system with a processer and video controller as well as input devices, all of which are also included in the Board Buddies system.

The above patents (for a remote gaming table, remote card game, and electronics converting chip game) suggest that Board Buddies can be patented. While there are many patented devices that can play remote games with physical pieces, many of them seem to revolve around casinos and card games. The Board Buddies system designed in this project is unique in that it plays Othello.

Detailed Technical Description of Project

The Board Buddies system is an IoT application that allows users to play a physical board game with each other despite being in different locations. The system consists of two physical boards, one for each player, that communicate with each other wirelessly.

Gameplay Logic

The Board Buddies game logic is based on the widely enjoyed two-player game of Othello [40] in which one player is assigned the white side while the other is assigned the black side of the dual-colored tiles. The goal of the game is to place or turn as many of the tiles to one’s color as

possible. The player with the most tiles on the board turned to their color when either the board is filled up entirely or there are no more possible moves is the winner.

The game begins with four tiles in the middle of the board, two black-side up and two white-side up. These tiles are arranged such that two tiles of the same color are diagonal from each other, in accordance with Figure 2. In order to have as many tiles of their color facing up as possible, a player must "outflank" their opponent's tiles. Outflanking occurs when two tiles of one color surround a series of tiles of the other color. The player with the surrounding tiles can flip over all the tiles between the two outer tiles to their color. The outflanking maneuver can be done horizontally, vertically, or diagonally (see Figure 3 for visual). The more tiles a player can surround in a single turn, the more they can flip to their color, which is the name of the game. Tiles can be surrounded from multiple directions in the same turn (e.g., both diagonally and horizontally).

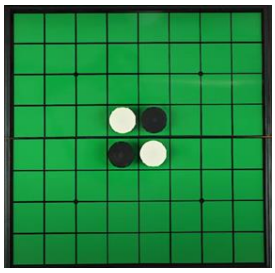


Figure 2 Othello Start State

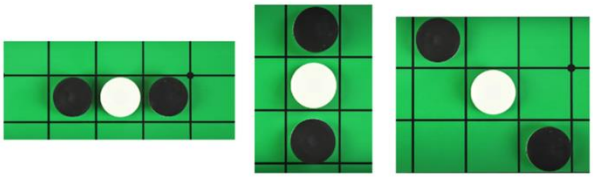


Figure 3 Valid Outflanking Maneuvers

Over the course of the contest, players go back and forth placing one tile each turn. During a turn, players must outflank at least one opponent tile and flip it to their color. If they are unable to do so, the player must forgo their turn until an opportunity arises where they can outflank their opponent. As mentioned, the game progresses until either the board is filled or neither player can outflank the other, at which point there are no moves left. When this stage is reached, players count the number of tiles with their color facing up, and whoever has more tiles is declared the winner.

Gameplay Implementation

We created a finite state machine to move the board through the various stages of the game. A diagram is seen in Figure 4.

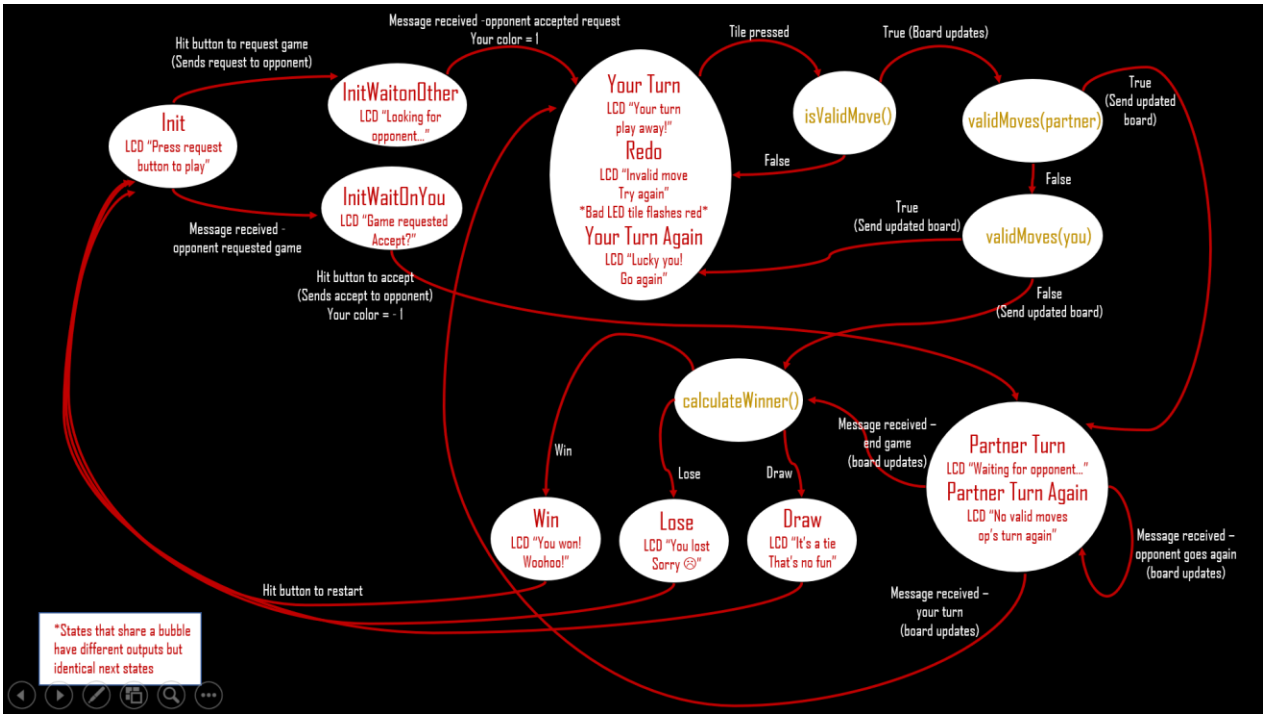


Figure 4 State Diagram

When the game is first powered on, it begins in the initial state (Init). A player gets out of this state when either the player presses the button to request a game (sent to InitWaitonOther) or a message is received saying that the other player has requested a game (sent to InitWaitonYou). To get out of InitWaitOnOther, the player must receive a message saying that the other player has accepted the game, which makes it the player's turn and gives the player the color "1" (which is blue in our design). To get out of InitWaitOnYou, the player must press the button to accept the game, which makes it the opponent's turn and gives this player the color "-1" (which is green in our design).

When it is a player's turn, the FSM looks for the user to press a tile. Once a single tile is pressed, the game algorithm first determines if that tile corresponds to a valid move for the player with isValidMove(). If it is not valid, the player is sent to the Redo state. If it is, isValidMove() will update the array holding the board state and the FSM calls validMoves() on the opponent. If the opponent has any valid moves, it becomes the opponent's turn. The opponent is sent a message indicating it is the opponent's turn and sharing the new board state, and the local player is sent to the PartnerTurn state. If the opponent does not have any valid moves, the FSM will check if the local player has any valid moves with validMoves(). If they do, the FSM will send a message indicating this player goes again and sharing the new board state, and the FSM will transition into YourTurnAgain. If they do not, it signifies the end of the game. The opponent will be sent a message with the updated board and an indication that the game is over. Then the FSM will call calculateWinner() to determine if the local player has won, lost, or tied the game. Then the FSM

will transition into the Win Lose or Draw state accordingly. Redo and YourTurnAgain are identical in next state logic to YourTurn, but they display a different message to the user.

In PartnerTurn, the player waits to receive a message from the other player indicating one of three things: it is this player's turn now, the opponent will go again, or we've reached the end of the game. The message will also contain an updated array for the state of the board which has any changes from the opponent's most recent move. The FSM will transition into YourTurn or PartnerTurnAgain or call calculateWinner() and transition to Win, Lose, or Draw accordingly. PartnerTurnAgain is identical in next state logic to PartnerTurn, but a different message is displayed to the player. Finally, Win, Lose, and Draw are all exited in the same way; pressing the button in any of these states resets the game and returns the player to Init.

The output function for each of these states is relatively simple, and entails sending a character to the LCD to display the right message for that state and sending signals to the LEDs to update the board colors. Redo is slightly more complicated because it first sends the LEDs a signal to light the board with the wrong tile the user just pressed (saved in the FSM) colored red, before sending the regular board state signal. This effectively causes the bad tile to flash red for a second, as a visual cue to the player.

The logic behind the three functions called by the FSM is also fairly straightforward. In Othello, a given move is valid if and only if it outflanks at least one of the opponent's tiles. To check if a move is valid, isValidMove() first checks that the tile pressed is indeed an unplayed spot on the board. If so, it then checks in all 8 directions to see if it can find any outflanked tiles. To do this, it first looks to see if the neighboring tile is of the opponent's color. If it is, it continues looking in that direction until edge of the board is reached or a blank tile is reached (no outflanking) or the player's color is reached (outflanking). If outflanked tiles are found, they are flipped to the player's color. Then isValidMoves() returns whether outflanked tiles were found and flipped (valid move) or no outflanked tiles were found (not a valid move).

To determine whether a given player has any valid moves, validMoves() calls a version of isValidMove() that does not flip tiles the outflanked tiles on, iterating through potentially all 64 locations of the board. As soon one call to isValidMove() determines there is at least one valid move, the iteration breaks and validMoves() returns true.

Because the players' colors are encoded as integers 1 and –1 in the array holding the game state (with 0 indicating blank), calculateWinner() takes a sum of all the tiles in the array. If the sign of the summation matches the sign of the player's colors, that player wins. If the sum is 0, it is a draw. Otherwise, the player loses.

Game Board User Interface

Figure 5 Final Mechanical Housing (left). CAD Model (right)

The board housing was CADed in Autodesk Inventor, in order to plan the mounts to integrate all PCBs and frames together. First, bolt the button PCBs onto a frame to position the buttons with even spacing in figure 6. The mechanical stability could be improved if these PCBs did not have any gaps between them, or if it were one PCB.

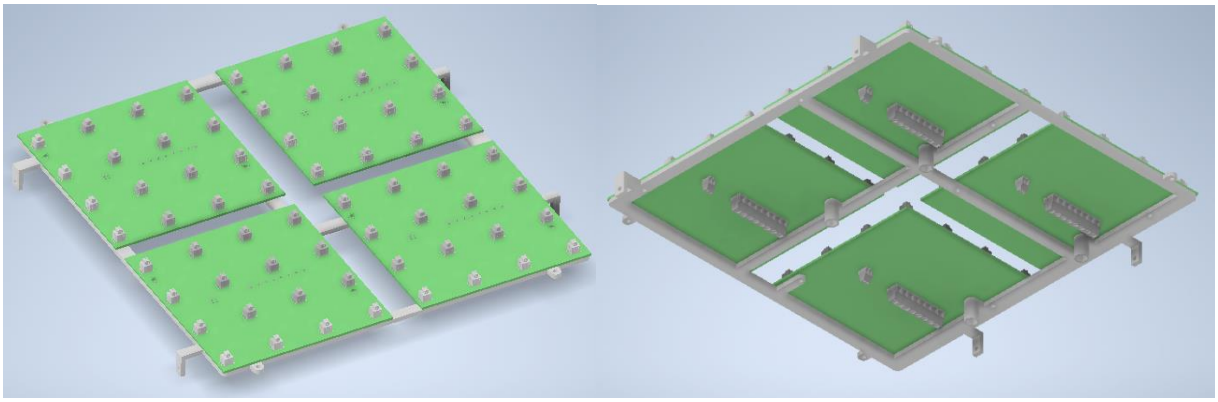


Figure 6 Button PCBs on Main Frame from Above and Below

When a button is pressed, that row and column will change from being pulled up to grounded. The button PCBs need to be connected by wires in the screw terminals to complete the row and columns.

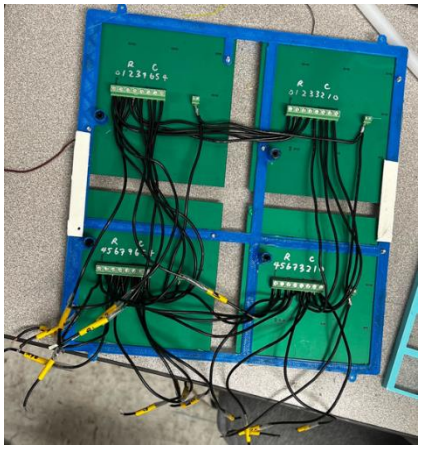


Figure 7 Button PCBs with Rows and Columns Connected

Then, shown in the figure below, the MCU PCB is screwed onto the frame's 4 built in standoffs.

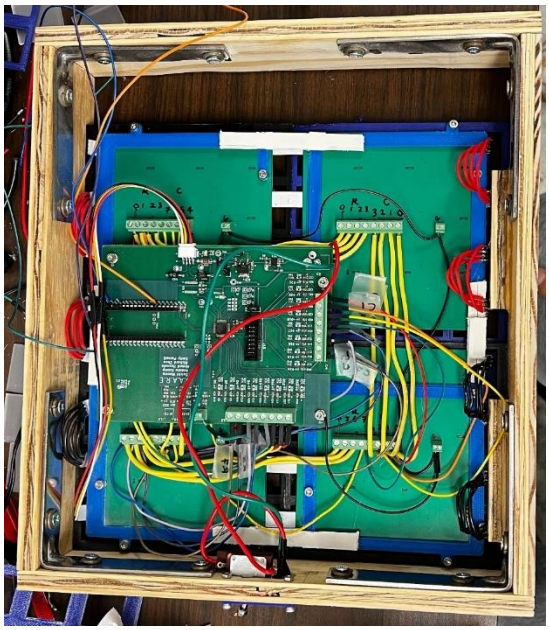


Figure 8 Assembled Board Back View

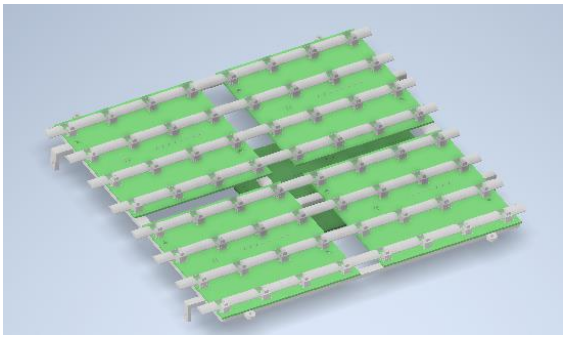


Figure 9 LED Strips Laid on Button PCBs

The roll of LED strip was cut into 8 sections, adhered to the PCBs and connected back together with wires. These are shown in figure 8 with red and black wires making arcs on the left and right sides. They form a string that starts from the top left corner and snakes down to the bottom left. The row and column wires screw into the MCU PCB's 16 terminals. The LCD Display [42] connects to the white connector on the MCU PCB. The power jack on the MCU was moved to the wooden frame with a switch in series with red and green wires.

A 3D printed grate is screwed onto the frame, shown below. Its purpose is to confine LED's light to its square. 3D printers on campus were sparse and not large enough to print the entire grate in one piece, so it was sectioned into quarters and reattached with glue afterwards. To meet mass production, 3D printed parts are designed to be able to be plastic injection molded using a 2-piece mold.

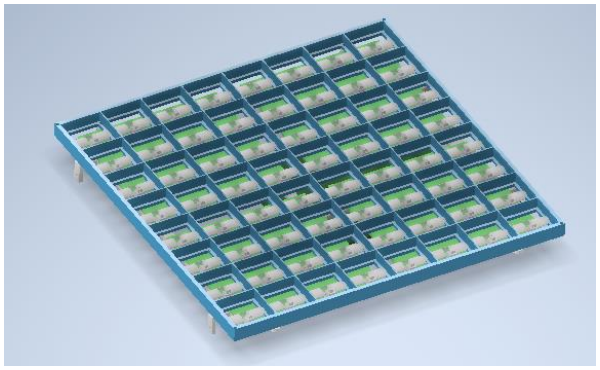


Figure 10 Grate on Frame

The assembly with grate slides into the wooden frame and the edges rest on wooden blocks glued to the sides of the wooden frame. Then, the 3D printed cover, with LCD display bolted in, is screwed on.

Each pushbutton stored within the board is connected to two wires. One wire tells us the row of the button, and the other tells us the column (if you are looking at the board as an 8x8 grid). In other words, the buttons are effectively multiplexed. This enabled us to reduce our number of button inputs from 64 to 16, ensuring that all the buttons could be directly connected to the MSP, rather than having to reduce the number of inputs through separate external devices. When called up by the gameplay FSM, functions that read in these button inputs will check through each of the 16 button inputs and check to see whether any of them have gone from 1 to 0. Buttons are active low, meaning when they are not being pressed, the inputs have a value of 1 whereas when a button is pressed, its row/column inputs go to 0. When certain inputs go to 0, the code analyzes which inputs have gone to 0 and calls upon a separate CovertButtonsToInputs() function to tell us exactly which button is being pressed (where it is located). From here the game logic analyzes the validity of the move that was enacted by pressing a given button and continues progressing.

The LEDs used in the user interface come in a strip. Each LED has a small integrated circuit that takes serial-peripheral interface (SPI) clock and data signals to determine the brightness and color of the LED. A chain of LEDs receives a start signal, a chain of commands for the LEDs, and an end signal. Each LED takes the first chunk of LED data for itself and only passes on the

LED data for the remaining LEDs. The function sendColors() was used to send color and brightness signals to the LEDs according to the current state of the board.

PCB Design

There are two sets of PCBs that were used in this project: the Microcontroller Unit (MCU) PCB and the button PCB. The MCU PCB is responsible for powering the MSP430FR2335 microcontroller as well as sending and receiving signals from the microcontroller. The high-level architecture of the PCB is shown below:

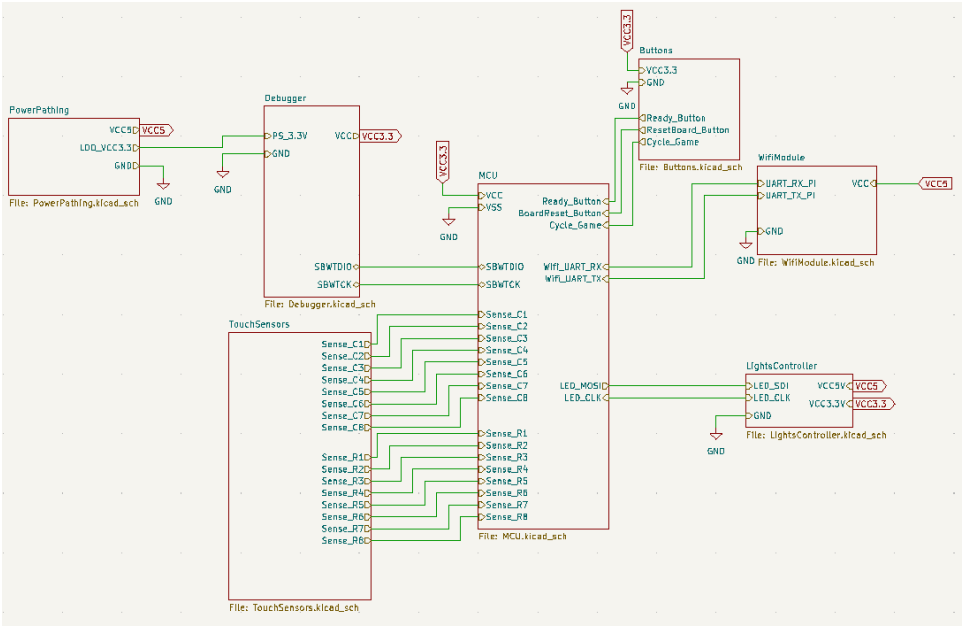


Figure 11 High Level Architecture of MCU PCB

The power supply circuitry of the MCU PCB is shown in Figure 11.

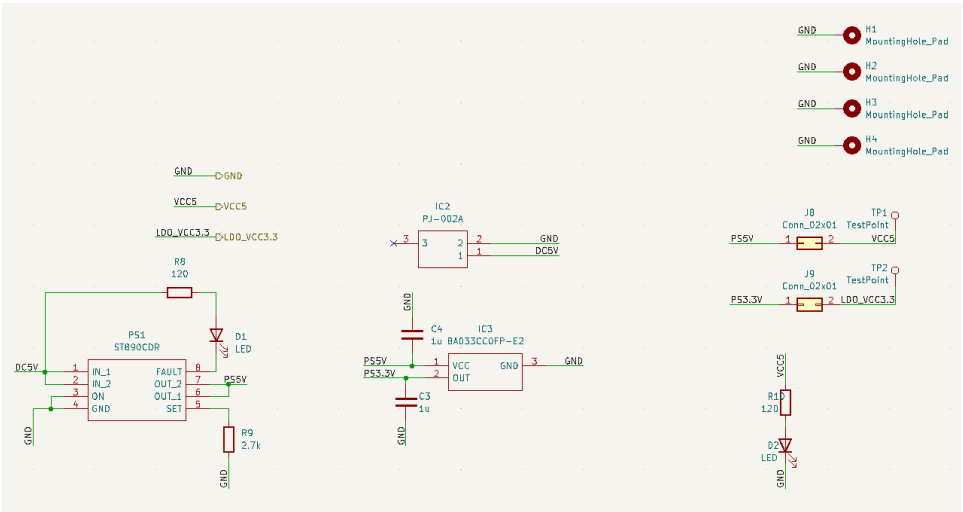


Figure 12 Power Supply Circuitry

As seen in the circuit, the entire device is powered from a PJ-002A jack [43] which connects to a 5V 2A AC to DC wall adapter. The 5V signal is then fed through the ST890CDR [44], which is an overcurrent protection high side switch that opens the 5V line if a current of 1.5A is reached. The current limited 5V output feeds into a 3.3V linear dropout (LDO) voltage regulator, which is the device powering the microcontroller.

Because the project utilizes a microcontroller chip rather than a development board, a debug interface was needed for sending code to the device. This circuitry is shown in Figure 13.

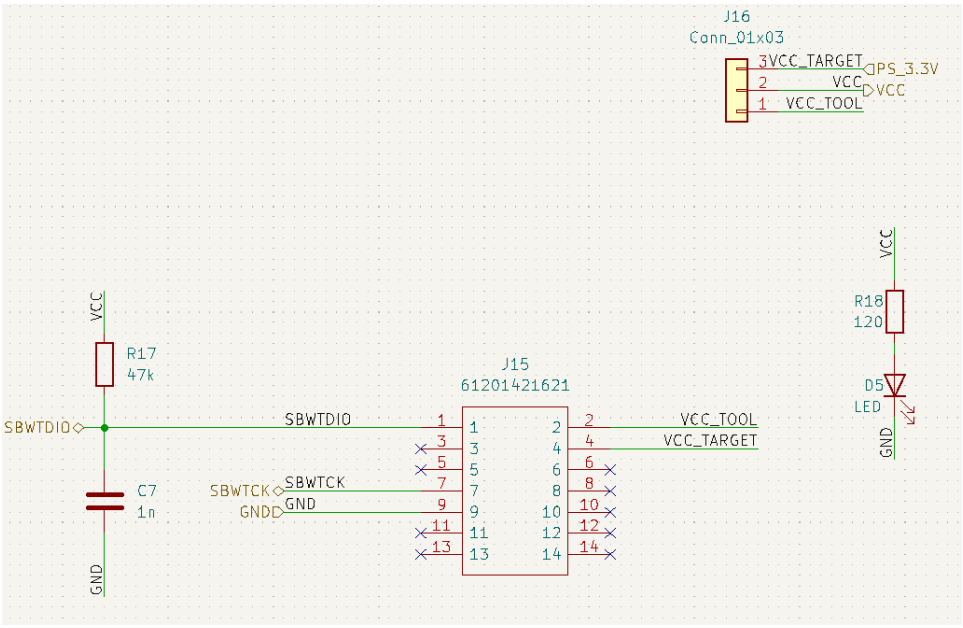


Figure 13 Debug Interface Circuitry

The interface chosen for debugging the MSP430 is the Spy-By-Wire interface as described by the MSP430FR2335 datasheet [18]. This interface requires the SBWTDIO and SBWTCK data lines. It also provides the option between power the MSP through the debug devices (such as another computer) or through an external power source (such as the power supply).

The rest of the data transfer circuitry was simple. For example, the Raspberry Pi Pico W used in this project communicates with the MSP430 through UART, which only requires that the UART_RX of one device connects to the UART_TX of the other devices and vice versa. The LEDs communicate to the MSP430 through a simplified version of SPI in which the LEDs have a data in line and a clock line which controls the strips.

Lastly, since the microcontroller must keep track of 16 button inputs, 16 GPIO inputs are used on the MSP430 to read the buttons. Externally, the buttons are switches with one end connected to ground and one end connected as an input to the microcontroller. The micro controller is programmed to read inputs with pull-up resistors, meaning that the input is active low. To protect the microcontroller, input protection is used as shown below.

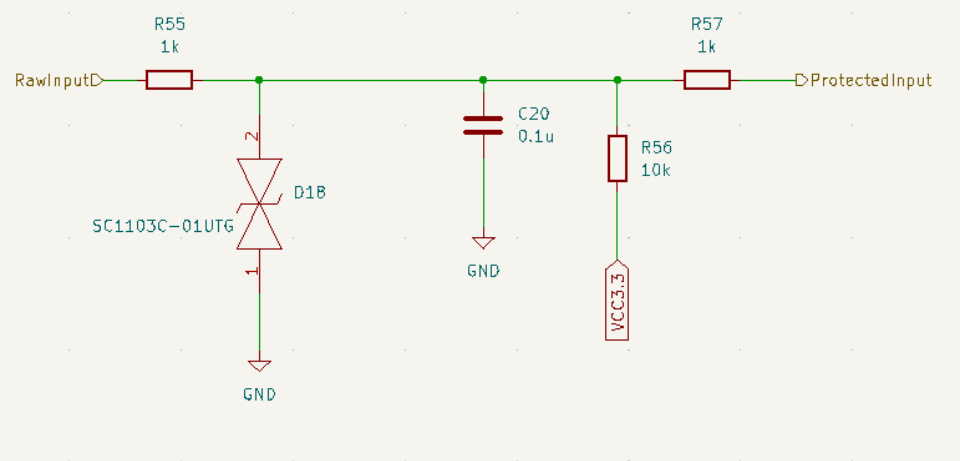


Figure 14 Input Protection Circuitry

The fully laid out PCB is shown below in Figure 15:

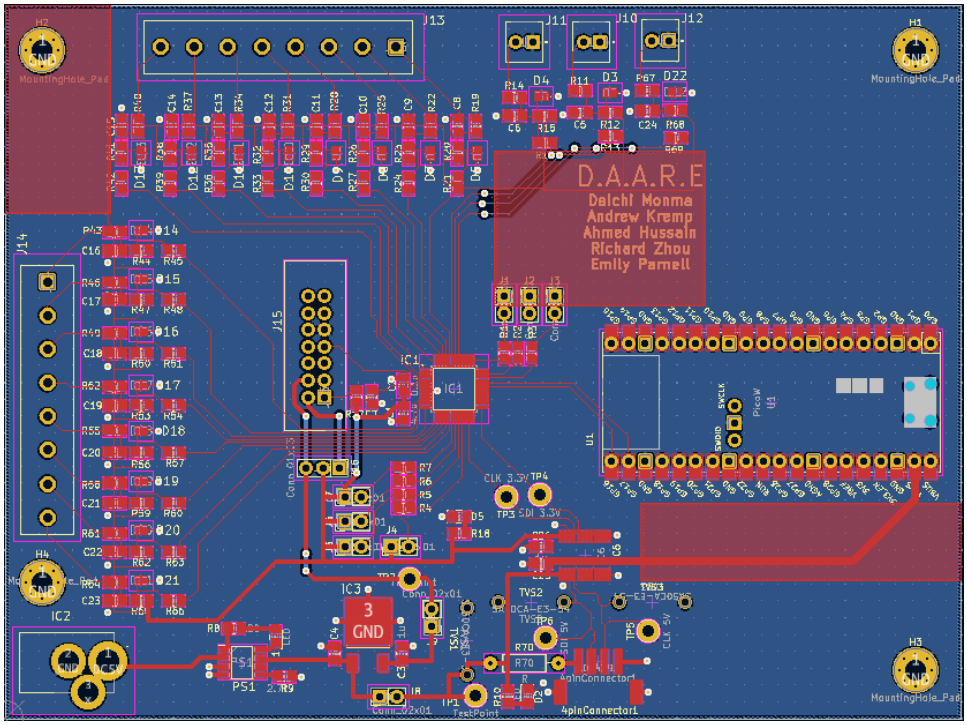


Figure 15 Final Layout of MCU PCB

One modification made to this circuit after it was fabricated and assembled was that the pullup resistor to VCC3.3 was desoldered and instead the pull-up resistor input option of the microcontroller was used instead. Also, during testing, the TVS diode (SC1103C-01UTG) was not functioning properly and created a resistance that interfered with the pull-up resistor reading of the microcontroller, so it was removed from the circuit as well.

The second set of PCBs that was designed was the button PCBs. The PCBs house the buttons of the physical board, where the buttons are evenly spaced to represent tiles. These buttons are all

tied to ground at one end and used as multiplexed inputs into the microcontroller. Since there are 64 possible buttons (8x8 grid), multiplexing was used to reduce the number of inputs. To accomplish this, a double pole button was used in which the button controls two separate lines simultaneously. Utilizing this, every button on the same row had one of the poles shorted together and every button on the same column had the other pole shorted. This means that anytime a column or row is pressed, the microcontroller would know. The button can then be identified using its column and row index.

Communication Between Boards

In order to get game state data between the two game boards wirelessly, we decided to enlist the help of two Raspberry Pi Pico W's as well as a central Linksys Wi-Fi router [45]. The two Picos are programmed using the MicroPython programming language, which is a variant of Python. Specifically, the Picos are programmed using the MicroPython "sockets" library to connect to and exchange data through the Linksys router's network.

The data being exchanged between the Picos primarily corresponds to the current state of the game. As mentioned earlier, our gameplay FSM is programmed using the MSP430FR2355 microcontroller. Thus, the MSP needed to be able to send new game state data as well as receive updated game state data from their corresponding Pico. The two devices communicated via a UART connection. For each MSP, its transmit buffer (TX) pin was connected to the receive buffer (RX) pin of a Pico and vice versa. Whenever a player makes a move, that player's MSP TX buffer is loaded with the updated game state. It then transmits that game state data to the Pico connected to it via UART. As mentioned, the Pico then sends the new game state data to the other Pico wirelessly through the Linksys router's network. Finally, said other board's Pico transmits the updated game state data to its MSP, again via UART. This sequence is shown visually via the diagram in Figure 16.

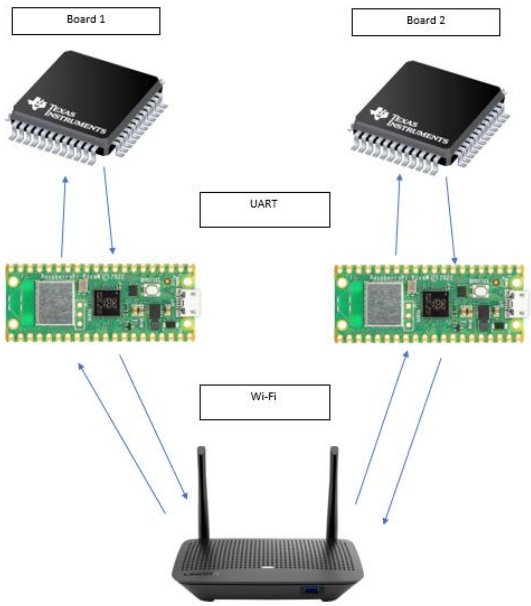


Figure 16 Communication Sequence

There is a function to check the MSP's RX buffer for updated game state messages, and once a board receives a message, the opponent's game board is updated, and it would be their turn to make a move.

In order to make debugging and testing this process easier, each Pico is connected to an LCD via an I2C adapter [46]. The LCD is programmed, along with the rest of the Pico code, in MicroPython and is configured to output a message depending on the current state of the gameplay FSM. For example, when it is someone's turn (i.e. they just received new game state data), the LCD displays a particular message to indicate that.

In addition to sending game state information back and forth, this communication sequence is also used for the request game/accept game functionality we implemented. Each board is configured with a request game/accept game button. When a user presses the request button, a message is sent to the other board indicating that their opponent has requested a game. Once the board receives said message, the request button becomes and accept button. That player then presses the accept game button, which sends a message to the other board that the opponent is ready to start the game. From there, the requesting player is slated to go first, and the game commences.

Modifications from Original Design

One modification from the original design involves swapping out the originally picked Wi-Fi System on a Chip (SOC) [47] module with Raspberry Pi Pico W's. It was determined that since none of us had ever programmed external modules via SPI from a microcontroller, which was required to set up these chips, it would not be smart to use this option. Given how important the wireless communication aspect is to the success of our project and how little time we must complete the project (about 2 months after the first round of parts came in), we decided it would be best to go with an option that has more clear documentation and can be accessed directly via USB.

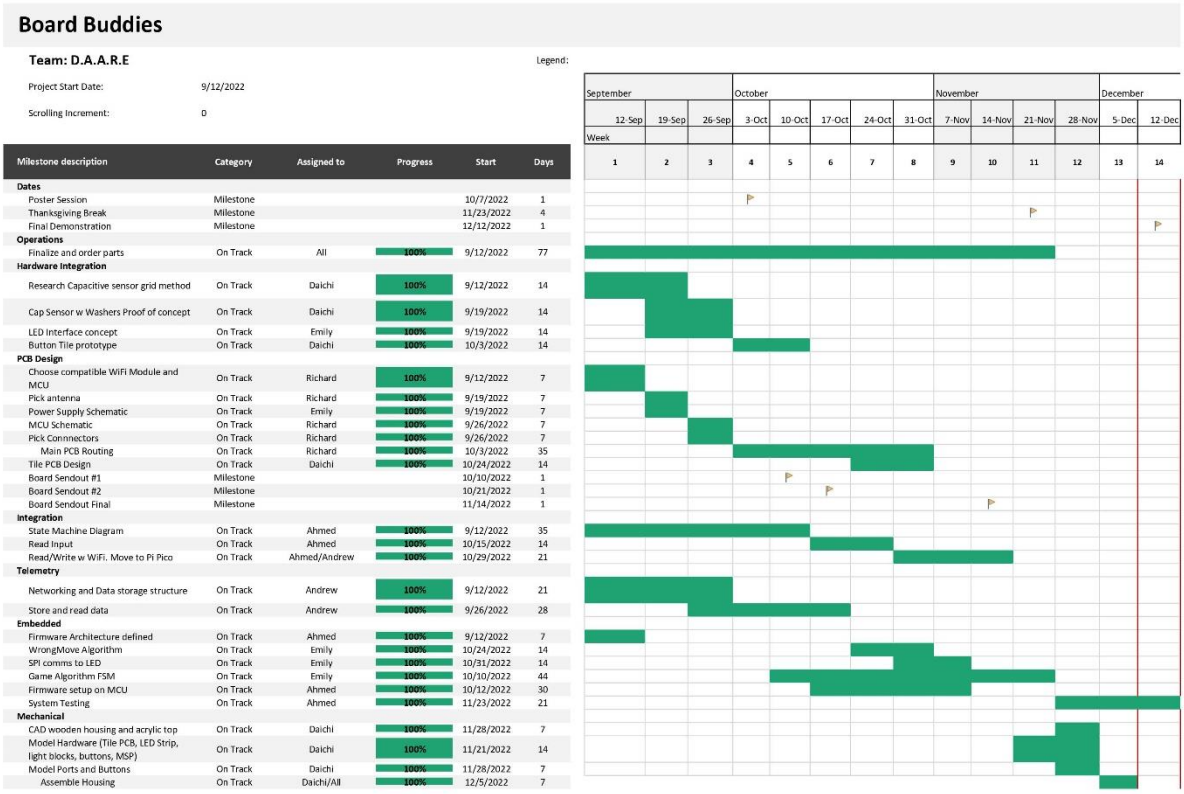
Similarly, the pushbutton inputs were not initially planned to be connected directly to the MSP430FR2355 chip. This was because we were under the assumption that we needed to save GPIO pins. Once we realized we would have plenty of pins on the MSP for the buttons, especially once we used the multiplexing system to reduce the number of button inputs for each board from 64 down to 16, we decided to route the buttons directly to the MSP rather than going through an external analog to digital converter or GPIO extender. This again saved the extra time it would have taken to program external modules.

Finally, the LED and LCD functionality were enhanced from our original design. For example, the LEDs now flash a tile red if a player makes an invalid move by touching that tile. Similarly, the LCD guides the player through every point of the game, rather than just declaring the winner/loser. Neither of those features were in the original design but were fitting updates for the result.

Project Timeline

The final project timeline changed significantly from the original across 2 months. At first glance of the Gantt charts below, ordering components was supposed to finish in 4 weeks, but continued until nearly the last chance in the semester. This was because mechanical design took longer than expected. CAD of the housing took longer than expected since every frame was custom designed and had to fit together. In the ideation phase, there were options that had more complex mechanical designs with moving parts, so picking the simplest option may have created a false sense of comfort. In reality, the mechanical side failed in prototyping which pushed back manufacturing the final housing. It failed because 3D printing tolerances are large and inconsistent.

On the scale of system testing, we allocated the last 3 weeks for running the whole device. If the housing was done by then, testing would have been easier and bugs could have been caught earlier.



Board Buddies

Team: D.A.A.R.E

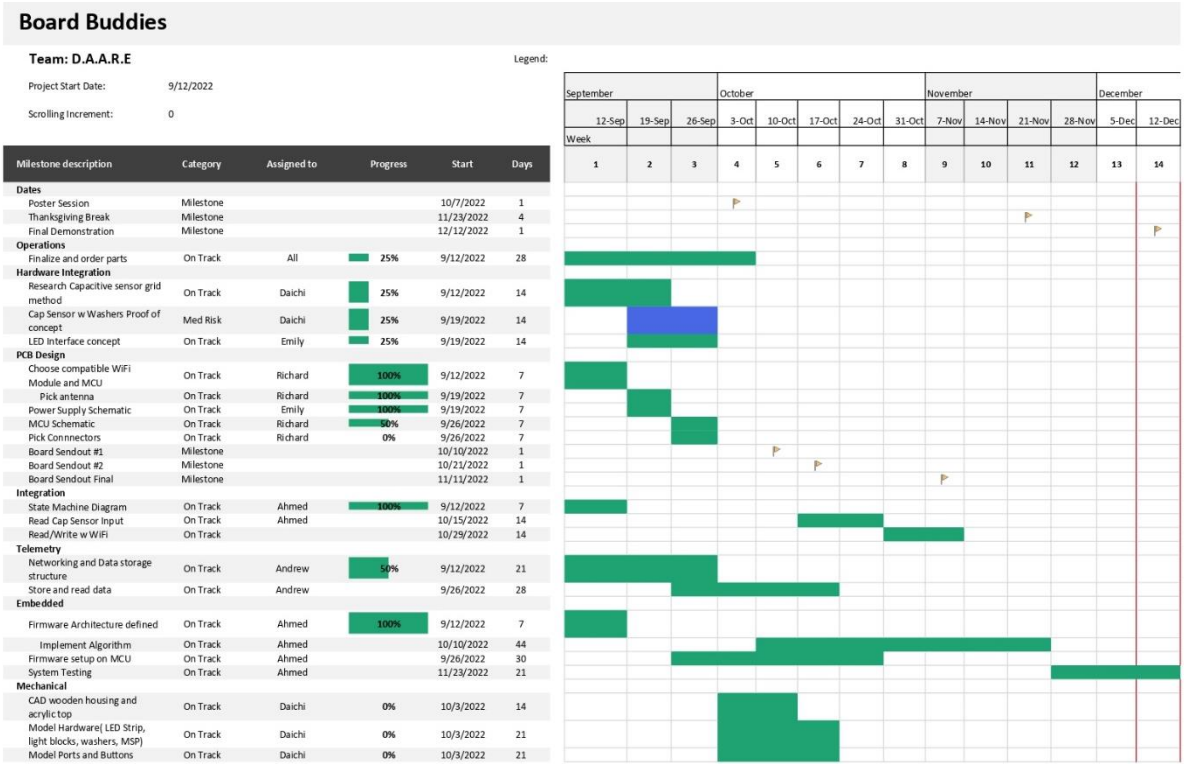
Project Start Date: 9/12/2022

Scrolling Increment: 0

Legend:

Milestone description	Category	Assigned to	Progress	Start	Days
Dates					
Poster Session	Milestone			10/7/2022	1
Thanksgiving Break	Milestone			11/23/2022	4
Final Demonstration	Milestone			12/12/2022	1
Operations					
Finalize and order parts	On Track	All	100%	9/12/2022	77
Hardware Integration					
Research Capacitive sensor grid method	On Track	Daichi	100%	9/12/2022	14
Cap Sensor w Washers Proof of concept	On Track	Daichi	100%	9/19/2022	14
LED Interface concept	On Track	Emily	100%	9/19/2022	14
Button Tile prototype	On Track	Daichi	100%	10/3/2022	14
PCB Design					
Choose compatible WiFi Module and MCU	On Track	Richard	100%	9/12/2022	7
Pick antenna	On Track	Richard	100%	9/19/2022	7
Power Supply Schematic	On Track	Emily	100%	9/19/2022	7
MCU Schematic	On Track	Richard	100%	9/26/2022	7
Pick Connnectors	On Track	Richard	100%	9/26/2022	7
Main PCB Routing	On Track	Richard	100%	10/3/2022	35
Tile PCB Design	On Track	Daichi	100%	10/24/2022	14
Board Sendout #1	Milestone			10/10/2022	1
Board Sendout #2	Milestone			10/21/2022	1
Board Sendout Final	Milestone			11/14/2022	1
Integration					
State Machine Diagram	On Track	Ahmed	100%	9/12/2022	35
Read Input	On Track	Ahmed	100%	10/15/2022	14
Read/Write w WiFi. Move to Pi Pico	On Track	Ahmed/Andrew	100%	10/29/2022	21
Telemetry					
Networking and Data storage structure	On Track	Andrew	100%	9/12/2022	21
Store and read data	On Track	Andrew	100%	9/26/2022	28
Embedded					
Firmware Architecture defined	On Track	Ahmed	100%	9/12/2022	7
WrongMove Algorithm	On Track	Emily	100%	10/24/2022	14
SPI comms to LED	On Track	Emily	100%	10/31/2022	14
Game Algorithm FSM	On Track	Emily	100%	10/10/2022	44
Firmware setup on MCU	On Track	Ahmed	100%	10/12/2022	30
System Testing	On Track	Ahmed	100%	11/23/2022	21
Mechanical					
CAD wooden housing and acrylic top	On Track	Daichi	100%	11/28/2022	7
Model Hardware (Tile PCB, LED Strip, light blocks, buttons, MSP)	On Track	Daichi	100%	11/21/2022	14
Model Ports and Buttons	On Track	Daichi	100%	11/28/2022	7
Assemble Housing	On Track	Daichi/All	100%	12/5/2022	7

Figure 17 Final Gantt Chart



Board Buddies

Team: D.A.A.R.E

Project Start Date: 9/12/2022

Scrolling Increment: 0

Legend:

Milestone description	Category	Assigned to	Progress	Start	Days
Dates					
Poster Session	Milestone			10/7/2022	1
Thanksgiving Break	Milestone			11/23/2022	4
Final Demonstration	Milestone			12/12/2022	1
Operations					
Finalize and order parts	On Track	All	25%	9/12/2022	28
Hardware Integration					
Research Capacitive sensor grid method	On Track	Daichi	25%	9/12/2022	14
Cap Sensor w Washers Proof of concept	Med Risk	Daichi	25%	9/19/2022	14
LED Interface concept	On Track	Emily	25%	9/19/2022	14
PCB Design					
Choose compatible WiFi Module and MCU	On Track	Richard	100%	9/12/2022	7
Pick antenna	On Track	Richard	100%	9/19/2022	7
Power Supply Schematic	On Track	Emily	100%	9/19/2022	7
MCU Schematic	On Track	Richard	50%	9/26/2022	7
Pick Connnectors	On Track	Richard	0%	9/26/2022	7
Board Sendout #1	Milestone			10/10/2022	1
Board Sendout #2	Milestone			10/21/2022	1
Board Sendout Final	Milestone			11/11/2022	1
Integration					
State Machine Diagram	On Track	Ahmed	100%	9/12/2022	7
Read Cap Sensor Input	On Track	Ahmed		10/15/2022	14
Read/Write w WiFi	On Track			10/29/2022	14
Telemetry					
Networking and Data storage structure	On Track	Andrew	50%	9/12/2022	21
Store and read data	On Track	Andrew		9/26/2022	28
Embedded					
Firmware Architecture defined	On Track	Ahmed	100%	9/12/2022	7
Implement Algorithm	On Track	Ahmed		10/10/2022	44
Firmware setup on MCU	On Track	Ahmed		9/26/2022	30
System Testing	On Track	Ahmed		11/23/2022	21
Mechanical					
CAD wooden housing and acrylic top	On Track	Daichi	0%	10/3/2022	14
Model Hardware(LED Strip, light blocks, washers, MSP)	On Track	Daichi	0%	10/3/2022	21
Model Ports and Buttons	On Track	Daichi	0%	10/3/2022	21

Figure 18 Original Gantt Chart from Proposal

Shown in the final Gantt chart above, Daichi's main roles were translating a person's touch into input to the MCU, as well as making the mechanical housing. These roles were independent from others, so it was easier to work in parallel. Andrew focused on network connectivity which depended on hardware with Richard because troubleshooting a problem involved determining whether it was hardware or software. Ahmed focused on software architecture and embedded programming; Richard led the PCB hardware; and Emily was mainly software lead that involved the LED control and state machine. It was important for someone to understand all the software even if they did not write it, because that helped narrow down issues during troubleshooting.

Test Plan

Figure 19 below shows the projected Test Plan from our proposal midway through the semester.

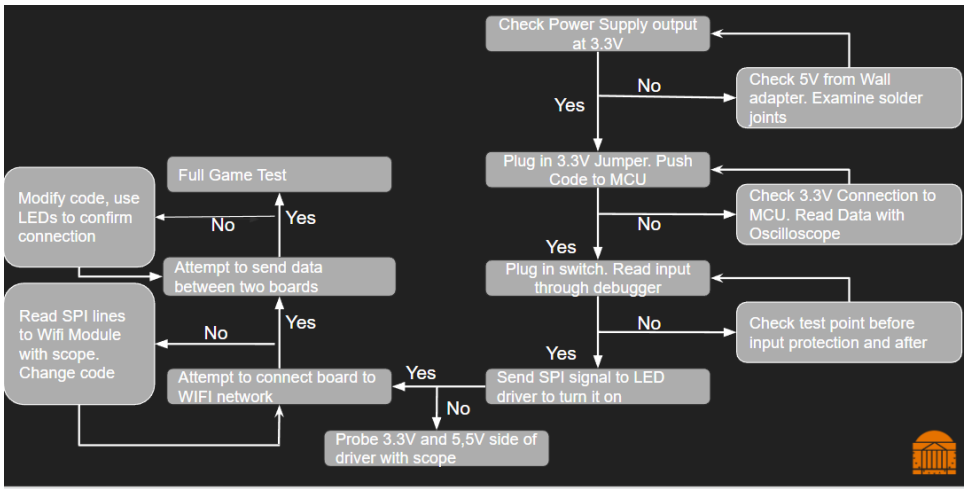


Figure 19 Proposal Test Plan

Although many of the individual submodule tests ended up being like what we anticipated in the proposal, the plan shown in Figure 19 is nowhere near comprehensive. As our design evolved, we had to add many more tests to the list.

Firstly, going into our proposal we were planning to use capacitive touch sensors rather than pushbuttons. However, after a few initial designs of these sensors proved futile, we figured that pushbuttons would be the way to go. Not only are they a little more straightforward to work with, but we figured users would enjoy the feeling of pushing a button rather than just touching a tile.

One thing we noticed almost immediately after the proposal was that we needed to test all the individual components that were being programmed before we put anything on a PCB. Thus, most of our preliminary tests were software-based. Since we designed our system to use an MSP430FR2355 chip rather than a launchpad, we couldn't do anything with that chip before we had our PCB. Thus, most initial software tests in CCS were done using a launchpad we recovered from a previous project. However, the chip on this launchpad was an MSP430F5529. Fortunately, despite some differences in the way registers were accessed, after some small adjustments the two MSPs functioned the same.

As such, we first tested trying to program our SOC Wi-Fi module through the launchpad using SPI. The goal was to connect this Wi-Fi module to a UVA Wi-Fi network. We quickly realized that this was going to be much more difficult than we anticipated. As mentioned in the "Detailed Technical Description of Project" section, given the limited time frame we had this semester, we made the decision to pivot towards using Raspberry Pi Pico W's, which had much clearer documentation than the Wi-Fi module, to connect to Wi-Fi. This test also showed us that there are some complications when trying to send data through UVA's Wi-Fi network, which caused us to pivot towards using a central Linksys Wi-Fi router to route our traffic through.

The next thing we tested was the logic in our gameplay FSM. Given that we didn't have many hardware components early in the semester, much of our time was spent developing the gameplay FSM initially. Once we had something to test, our first demonstration showed that we

could detect whether a move was valid using our game logic functions. This test was conducted using a 4x4 grid of buttons, a green LED (to indicate a valid move), and a red LED (to indicate an invalid move). At this point, there was no wireless communication set up, and we only had one launchpad. Thus, we could only test one move at a time, but initial signs pointed to the game logic working properly.

The next testing involved trying to turn on our LEDs, again using the launchpad. Once we determined that they could turn on properly, we integrated them with the gameplay logic to see if they would change color based on a given move. Fortunately, they did change color as expected.

After this, we tested whether the Raspberry Pi Pico W devices could connect to our central router. This test was conducted without any MSP interaction. Once it was determined that the Picos could indeed connect to the central router, we had to come up with a way to get game state data from the MSPs to the Picos.

Initially, we were planning to use SPI to communicate between the MSPs and Picos but ultimately decided that UART communication would be best for our purposes. Both the MSP430F5529 and MSP430FR2355 have UART capabilities, which was ideal. As such, our next test involved connecting the launchpad to one of the Picos' UART pins and trying to transmit/receive a message through UART. We started by trying to transmit/receive a single character, and once this worked, we tested functions that could transmit/receive strings. The string we sent back and forth was of course "Hello World".

After all these preliminary tests were completed, our design was pretty much set in stone, and, as such, we started adding components to our PCBs. The first components to get added to the board dealt with the power supply circuitry. Thus, the next thing to get tested was whether we could power the board using external power (a wall outlet). Once this worked, we tested UART communication between a MSP430FR2355 chip (soldered onto one of our PCBs) and a Pico. After some tweaks to optimize the code, this worked similarly to how it did with the launchpad.

We then decided to try and turn on the LEDs using the MSP on the PCB, which didn't work as expected. We also tried to read pushbutton inputs from the pushbutton PCB. Both tests had some frustrating complications. For the buttons, we made sure to some hardware tests to make sure they were connected properly, which they were. Finally, after checking the GPIO registers we were trying to access, we realized that they weren't working as expected (basically, all the pin values were random). It turns out that the MSP430FR2355 chip has an additional line of code required to activate GPIO pins that is not required for most MSP microcontrollers. The line is shown in Figure 20 below.

```
PM5CTL0 &= ~LOCKLPM5;          // Disable the GPIO power-on default high-impedance mode
                               // to activate previously configured port settings
```

Figure 20 GPIO Activation Code MSP430FR2355

Once this line was added to our initialization functions, the LEDs turned on, and the pushbuttons became readable. However, there was some inconsistency in the pushbutton readings, which we

found out was due to some excess solder residue shorting certain buttons to ground. Once this problem was fixed, the buttons worked as intended.

At this point, everything was set up and working on one PCB, so we planned to conduct a full game test with one “board” controlled by the PCB and the other controlled by the launchpad. However, before doing so, we decided to configure two LCDs (one for each board) to help us debug the game FSM. Basically, as mentioned previously, LCD would send a unique message depending on what state the program was in.

Once the LCDs were hooked up, a full game test was conducted. This revealed some slight issues with the gameplay logic that we hadn’t seen before, which were quickly rectified. However, we did get to the point where we could complete a full game without any unexpected failures.

Thus, we put together the other gameplay PCB, conducting the same tests that were conducted for the other gameplay PCB along the way. Once it was working, we then conducted the same full game test to make sure the launchpad wasn’t hiding an issue.

After we concluded the gameplay FSM worked as intended, we then integrated the LEDs to make sure they lit up appropriately over the course of the game. This caused us to realize that certain colors work better than others over the course of a game, and we adjusted the colors accordingly.

Lastly, all the PCBs and other components were placed inside the board housings, and a series of full games were played using the actual game boards. Once it was determined that the gameplay and different user interfaces worked as desired, testing was complete.

Final Results

After all the storms passed through and the dust settled, we produced two game boards just as described. To refresh, the criteria we proposed for the final Board Buddies system can be seen in Figure 21. I will go through our execution of each in depth below.

1. There are 2 physical boards
2. Players can push a button to power on the boards, initializing the tiles, or a button to return the game to its initial state
3. A tile lights up the appropriate color when a player taps it
4. Outflanked tiles flip change color in response to a player’s move
5. One player’s move shows up on the opponent’s board
6. The player cannot make another move until the opponent has gone
7. When the game ends, the players are notified who won on LCD

Figure 21 Proposed Design Requirements

Regarding the first proposed requirement, we do indeed have two physical boards (seen in Figure 22). The board housings are made from plywood, and each board contains an 8x8 grid of pushbutton tiles. Within each tile is a pushbutton as well as an LED to indicate the current color of the tile.

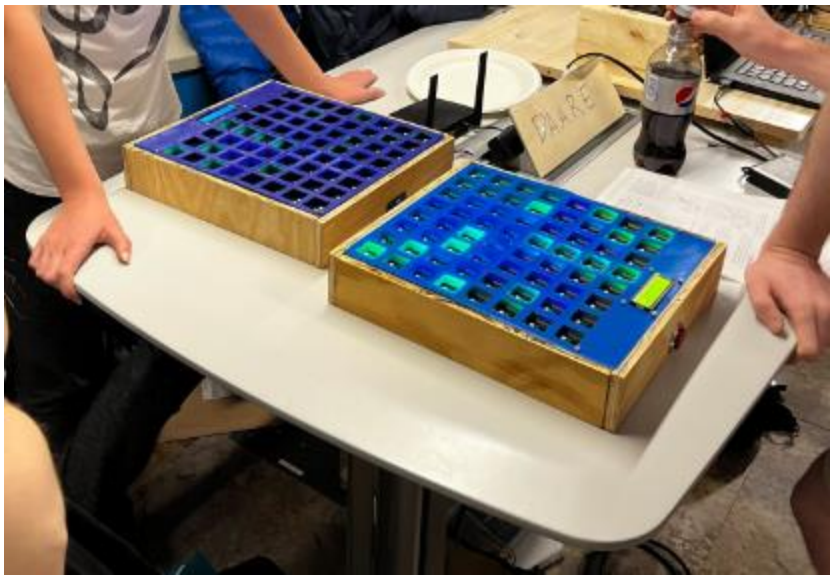


Figure 22 Playing a Game of Othello on Demo Day

The button laid out in requirement two has been changed to a switch, but it still functions as described. When the switch is toggled on, the board is powered on (via a wall outlet), putting the game in its initial state. In this state, the LCD display prompts the user to "Press Request Button to Start Game." To exit this state, users can press the designated request game/accept game button to begin their game. The LCD will update accordingly. When the switch is turned off, the board is powered off. If the switch is toggled again, the board is powered back on, effectively resetting the board back to its initial state.

Speaking about the request game/accept game functionality, we put in an additional button that functions initially as a request game button. It is described in detail in the "Detailed Technical Description of Project" section, but, basically, when one player presses the button, it sends a request to the other person's board. The LCD of the player who is requesting updates to "Looking for Opponent..." when they press the button. When a board receives a request message, their LCD updates to "Game Requested. Accept?". At this point, the request button functions as the button to accept the game. When the accepting player presses the accept button, the game commences. We added this feature to show that, should this board be sold commercially, and multiple games be programmed into the microcontroller, we could modify the communication message such that users could request the game they wanted using this button.

The third through fifth requirements involve the integration of the LED logic within the gameplay FSM. Fortunately, this does work as desired. When one player's LCD displays the message "Your Turn. Play Away.", that player is free to make a move. If the player makes an invalid move, the tile they pressed flashes red, and the LCD tells the player "Invalid Move. Try Again." If the player makes a valid move, all the tiles they outflank turn to their color. The updated board state is then sent to the other player, and that player's board updates to the current board state. Their LCD then updates to "Your Turn. Play Away." at which point they are free to make their move.

For the player who is not currently making a move, their LCD reads “Waiting for Opponent...”. This player can try pressing any buttons they want, but nothing will happen until it is their turn. This fulfills requirement six as shown in Figure 21.

There are a few additional features to the game that were not mentioned in the requirements. One was mentioned with the request/accept game button. Another feature includes handling what happens if a player gets to go twice. This happens when a player makes a move, and their opponent no longer has any valid moves. In such a situation, the player who just made a move gets to go again per the rules of Othello.

Thus, when this happens, the LCD of the player who just made a move shows “Lucky You! Go Again.”, and the LCD of the opponent who doesn’t have any valid moves now updates to “No Valid Moves. Opp’s Turn Again.”. The person who gets to go again can make their next move, and, if it opens a valid move for the opponent, the game continues as normal. Otherwise, the player keeps getting the same LCD message until the opponent has a valid move or the game ends.

Speaking of the game ending, requirement seven says that we will indicate whoever won using the LCDs. This is indeed the case. If a player wins, their LCD displays “You WON!!! WOOHOO”. If a player loses, their LCD displays “You Lost. Sorry :(". If the players tie, both of their LCDs show the message “It’s a tie. That’s no fun!”.

Overall, our design has adhered to all the criteria we laid out and then some. With the Board Buddies system, users can play a full game of Othello wirelessly just as we proposed. All the features work as described, and users should clearly understand what’s going on at every point of the game.

Costs

The detailed costs of the project components are listed on the spreadsheet in the appendix (under List of Materials Cost). The total of the $500 budget that was spent is $418.01. Another $31.66 was spent on manufacturing the tile PCBs from JLC PCB (which was chosen to reduce the cost of manufacturing large PCBs). There were also miscellaneous materials that were already available or extra spare components that were not purchased using the $500 budget. Figure 23 shows a rough estimate of a single set of Board Buddies Boards (2 physical boards) as well as when manufactured at 10000 units. The 10000 units values are estimated by looking through digikey for high quantity costs of components.

Cost of 1 Set of Board Buddies (2 Physical Boards)		
	Cost for 1 Unit	Cost per unit for 10000 units
MCU PCBs	$66	$2.41
Tile PCBs	$31.66	$7.19
PCB Components	$118.95	~$26.312
Raspberry PI Pico W	$25.98	$12
LEDs	$39.99	$39.99
Buttons	$47.84	$28.16
Total	$283	$89.75

Figure 23 Estimated Cost

While the high volume of manufacturing drastically reduces the cost of the PCB s and PCB components, the LEDs and buttons still are expensive. The cost of the LEDs might be able to be reduced by designing our own LED communication system and integrating it into the tile PCBs rather than purchasing them from a 3rd party. The same could be done for the buttons. The device created in this project had no assembly cost since it was completely assembled by members of the team. In the case of high-volume production this would not be possible, but costs would be relatively low due to automated assembly machines.

Future Work

There are several ways to expand upon the project we delivered. Perhaps the most obvious way would be to add another game to the design. There are many games, such as checkers, which also utilize 2 colors on an 8x8 board. This would require changing the initial states of the FSM to account for cycling through different game choices. Another thing would be to add functionality to see a player's most recently made move. Perhaps a player goes away from the board and misses where the opponent just played. A last-move button may be used to flash the most recently played tile to catch the player up on what happened. Finally, we designed our two boards to only be able to connect with each other, symbolizing a special connection between the players like a grandparent and grandchild or significant others. However, several beta testers indicated an interest in a game board which can connect to any other game board, not just its partner board. This functionality might be more applicable to someone who has several friends and wants to be able to play with them all, not only one friend who possesses the partner board. One further rendition of this project might be to give the boards this functionality to connect with any desired board, instead of a single partner board.

References

[1] "What is IoT (Internet of Things)? | Microsoft Azure," *azure.microsoft.com*. https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iot/#overview

[2] J. Gelinas, "Digital Fatigue: How to Make Technology Work for You, Not Against You," *TalentCulture*, Mar. 17, 2021. https://talentculture.com/digital-fatigue-how-to-make-technology-work-for-you-not-against-you/

[3] “Board Games,” *poki.com*. https://poki.com/en/board (accessed Sep. 12, 2022).

[4] “Play the best free online board games & puzzle games,” *playpager.com*. https://playpager.com/ (accessed Sep. 12, 2022).

[5] “| Nintendo - Customer Service | Game Boy Advance - Universal Link Cable Hookup Help.” https://www.nintendo.com/consumer/systems/gameboyadvance/hook_universallink.jsp (accessed Sep. 12, 2022).

[6] “| Nintendo - Customer Service | Game Boy Advance - Wireless Adapter FAQ.” https://www.nintendo.com/consumer/systems/gameboyadvance/agb_wireless_faq.jsp (accessed Sep. 12, 2022).

[7] M. Rosenfield, “Computer vision syndrome (aka digital eye strain),” Optometry in practice, vol. 17, no. 1, pp. 1–10, 2016.

[8] L. Hale and S. Guan, “Screen time and sleep among school-aged children and adolescents: a systematic literature review,” Sleep medicine reviews, vol. 21, pp. 50--58, 2015.

[9] “What is a Printed Circuit Board (PCB)?,” *Printed Circuits LLC*. https://www.printedcircuits.com/what-is-a-pcb/

[10] “Solar Car Team at UVA,” *www.solarcaratuva.org*. https://www.solarcaratuva.org/ (accessed Sep. 27, 2022).

[11] “Gizmologists at UVA,” *www.gizmologists.com*. http://www.gizmologists.com/ (accessed Sep. 27, 2022).

[12] “Cavalier Autonomous Racing,” *Cavalier Autonomous Racing*. https://autonomousracing.dev/ (accessed Sep. 27, 2022).

[13] Tesla, “Electric Cars, Solar Panels & Clean Energy Storage | Tesla,” *Tesla.com*, 2022. https://www.tesla.com/

[14] “Home | www.caltech.edu,” *Caltech.edu*, 2019. https://www.caltech.edu/

[15] *KiCad*. [Online]. Available: https://www.kicad.org/

[16] “Bluefin innovations,” *www.blfinn.com*. https://www.blfinn.com/ (accessed Sep. 27, 2022).

[17] “Welcome to Jacobs,” *Jacobs*, 2019. https://www.jacobs.com/

[18] “MSP430FR235x, MSP430FR215x Mixed-Signal Microcontrollers 1 Device Overview.” Accessed: Dec. 14, 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/msp430fr2355.pdf?ts=1670950036420

[19] “About Texas Instruments | TI.com,” *Ti.com*, 2020. https://www.ti.com/about-ti/company/overview.html

[20] “CC3200 SimpleLink™ Wi-Fi ® and Internet-of-Things Solution, a Single-Chip Wireless MCU 1 Device Overview 1.1 Features 1 Programming Interfaces (APIs) • CC3200 SimpleLink Wi-Fi-Consists of Applications Microcontroller, Wi-Fi Network • 8 Simultaneous TCP or UDP Sockets Processor, and Power-Management Subsystems • 2 Simultaneous TLS and SSL Sockets

• Wi-Fi CERTIFIED™ Chip -Powerful Crypto Engine for Fast, Secure Wi-Fi • Applications Microcontroller Subsystem and Internet Connections with 256-Bit AES Encryption for TLS and SSL Connections -ARM ® Cortex ® -M4 Core at 80 MHz -Station, AP, and Wi-Fi Direct ® Modes -Embedded Memory -WPA2 Personal and Enterprise Security • RAM (Up to 256KB) - SimpleLink Connection Manager for • External Serial Flash Bootloader, and Autonomous and Fast Wi-Fi Connections Peripheral Drivers in ROM -SmartConfig™ Technology, AP Mode, and -32-Channel Direct Memory Access (µDMA) WPS2 for Easy and Flexible Wi-Fi Provisioning - Hardware Crypto Engine for Advanced Fast -TX Power Security, Including • 18.0 dBm @ 1 DSSS • AES, DES, and 3DES • 14.5 dBm @ 54 OFDM • SHA2 and MD5 -RX Sensitivity • CRC and Checksum • -95.7 dBm @ 1 DSSS -8-Bit Parallel Camera Interface • -74.0 dBm @ 54 OFDM -1 Multichannel Audio Serial Port (McASP) Interface with Support for Two I2S Channels -Application Throughput -1 SD/MMC Interface • UDP: 16 Mbps -2 Universal Asynchronous Receivers and • TCP: 13 Mbps Transmitters (UARTs) • Power-Management Subsystem -1 Serial Peripheral Interface (SPI) -Integrated DC-DC Supports a Wide Range of -1 Inter-Integrated Circuit (I 2," 2013. Accessed: Dec. 14, 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/cc3200.pdf?ts=1670985161865&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC3200

[21] "Raspberry Pi Documentation - Raspberry Pi Pico," *www.raspberrypi.com*. https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-Pico.html

[22] "PCB Prototype & PCB Fabrication Manufacturer - JLCPCB," *jlcpcb.com*. https://jlcpcb.com/VBS?utm_source=bing_vbs&utm_medium=cpc&utm_campaign=422890460&utm_content=&utm_term=e_jlcpcb&adgroupid=1345803553234414&msclkid=f45d3d207e321ce963fc5eb94970ece9 (accessed Dec. 14, 2022).

[23] "MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers 1 Features," 2009. Accessed: Dec. 14, 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/msp430f5529.pdf?ts=1670956045554&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430F5529

[24] ["ELECTROSTATIC SENSITIVE DEVICES SK9822 SPECIFICATION INTEGRATED LIGHT SOURCE INTELLIGENT CONTROL (Double line transmission) OF CHIP-ON-TOP SMD TYPE LED." Accessed: Dec. 14, 2022. [Online]. Available: https://www.pololu.com/file/0J1234/sk9822_datasheet.pdf

[25] "CCSTUDIO Code Composer Studio (CCS) Integrated Development Environment (IDE) | TI.com," *Ti.com*, 2020. https://www.ti.com/tool/CCSTUDIO

[26] "Thonny, Python IDE for beginners," *Thonny.org*, 2018. https://thonny.org/

[27] "KiCad EDA," *www.kicad.org*. https://www.kicad.org/

[28] "Inventor Software | Get Prices & Buy Official Inventor 2023 | Autodesk," *Autodesk.com*, Jan. 08, 2021. https://www.autodesk.com/products/inventor/overview?plc=INVPROSA&term=1-YEAR&support=ADVANCED&quantity=1&tab=subscription (accessed Dec. 14, 2022).

[29] “What is PLA Plastic (Polylactide)? | A Simple and Definitive Guide,” *PlasticRanger*, Dec. 09, 2021. https://plasticranger.com/what-is-pla-plastic/

[30] “What is ABS Plastic | How is ABS Material Made | Key Properties of ABS | Applications of ABS Plastic | Advantages & Disadvantages of ABS Plastic,” *PlasticRanger*, Aug. 11, 2021. https://plasticranger.com/what-is-abs-plastic/

[31] NIOSH [2020]. 3D printing with filaments: Health and safety questions to ask. By Glassford E. Dunn KL, Dunn KH, Hammond D, Tyrawski J. Cincinnati, OH: US Department of Health and Human Services, Centers for Disease Control and Prevention, National Institute for Occupational Safety and Health, DHHS (NIOSH) Publication No. 2020-115, https://doi.org/10.26616/NIOSHPUB2020115 (accessed Sep 12, 2022)

[32] CDC. “3D Printing Safety at Work”. https://www.cdc.gov/niosh/newsroom/feature/3Dprinting.html (accessed Sep 12, 2022)

[33] NEMA. “About NEMA Standards”. https://www.nema.org/standards/about-standards (accessed Sep 12, 2022)

[34] NEMA. “NEMA Enclosure Ratings”. https://www.integraenclosures.com/resources/nema-enclosure-types-3x-4x/ (accessed Sep 12, 2022)

[35] IEEE. “IEEE 802.11 Wireless Local Area Networks”. https://www.ieee802.org/11/ (accessed Sep 12, 2022)

[36] IEC. “IEC 60950-1:2001”. https://www.iecee.org/dyn/www/f?p=106:49:0::::FSP_STD_ID:18568 (accessed Sep 12, 2022)

[37] J. Chun, “Remote live table gaming terminals and systems,” US20160012674A1, Jan. 14, 2016

[38] G. Makhoul, J. Rosemeyer, and A. Brimer, “Game systems and methods for remote card games using physical playing cards,” US20100105460A1, Apr. 29, 2010

[39] A. Baerlocher, W. Bussick, and M. MacVittie, “Gaming device having board and converting chip game,” US20050054405A1

[40] “Othello Game - Play Othello online,” *Othello*. https://www.othelloonline.org/ (accessed Sep. 27, 2022).

[41] “How to Play Othello” https://www.youtube.com/watch?v=zFrlu3E18BA (accessed Sep 12, 2022)

[42] “LCD-1602A Datasheet PDF - Datasheet4U.com,” *datasheet4u.com*. https://datasheet4u.com/datasheet-pdf/CA/LCD-1602A/pdf.php?id=519148 (accessed Dec. 14, 2022).

[43] “PJ-002A Datasheet PDF” cuidevices.com. https://www.cuidevices.com/product/resource/pj-002a.pdf (accessed Dec. 14, 2022).

[44] “ST890 1.2 A current limited high-side power switch with thermal shutdown Datasheet - production data,” 2016. Accessed: Dec. 14, 2022. [Online]. Available:

https://www.st.com/content/ccc/resource/technical/document/datasheet/1a/8c/51/d8/96/28/4d/63/CD00002878.pdf/files/CD00002878.pdf/jcr:content/translations/en.CD00002878.pdf

[45] “Linksys WiFi 5 Router Dual-Band AC1200 (E5400),” *Linksys: US*. https://www.linksys.com/linksys-wifi-5-router-dual-band-ac1200-e5400/E5400.html (accessed Dec. 14, 2022).

[46] “PCF8574,” *Ti.com*, 2016. https://www.ti.com/product/PCF8574 (accessed Dec. 14, 2022).

[47] Mouser, “Microchip ATWINC1510-MR210UB,” *Mouser Electronics*, 2022. https://www.mouser.com/ProductDetail/Microchip-Technology-Atmel/ATWINC1510-MR210UB?qs=QB0ovJDgFgt5QuIkS9qN8w%3D%3D&mgh=1&gclid=CjwKCAjwpqCZBhAbEiwAa7pXeW3SQivL_exQpKcmyOzyLjuMbVtWmRGzJP6et5ANC5BAUOpWN7vFLRoCQGkQAvD_BwE (accessed Sep. 20, 2022).

Appendix

List of Material Costs:

Item Number		Qty	Unit Cost	Total
1	BTF-LIGHTING SK9822 - 5m, 30 LEDs/m	1	$ 39.99	$ 39.99
2	Adafruit - ID 4045	2	$ 0.95	$ 1.90
3	Adafruit - ID 4392	1	$ 3.50	$ 3.50
4	MSP430FR2355TPTR	2	$ 4.20	$ 8.40
5	SN74LVC4245AIPWREP	2	$ 3.52	$ 7.04
6	ATWINC1510-MR210UB	2	$ 11.27	$ 22.54
7	LM1117MPX-33NOPB	2	$ 0.59	$ 1.18
8	ST890CDR	2	$ 2.97	$ 5.94
9	SA5.0CA-E3/54	6	$ 0.57	$ 3.42
10	PH1-03-UA	2	$ 0.10	$ 0.20
11	1729186	8	$ 5.29	$ 42.32
12	1725656	8	$ 2.00	$ 16.00
13	22284028	18	$ 0.29	$ 5.22
14	PJ-002A	2	$ 0.71	$ 1.42
15	SC1103C-01UTG	40	$ 0.51	$ 20.48
16	CRCW080510K0FKEAC	60	$ 0.06	$ 3.60
17	CRCW0805120RFKEAC	6	$ 0.10	$ 0.60
18	CR0805-FX-1001ELF	80	$ 0.10	$ 8.00
19	RC0805FR-101ML	4	$ 0.10	$ 0.40
20	CR0805-JW-473ELF	2	$ 0.10	$ 0.20
21	5001	20	$ 0.42	$ 8.40
22	CRCW08052K70FKEAC	2	$ 0.10	$ 0.20

Figure 24 Bill of Materials Part 1

22	CRCW08052K70FKEAC	2	$ 0.10	$ 0.20
23	C0805C104M5RACAUTO	40	$ 0.14	$ 5.60
24	CC0805KKX5R7BB475	2	$ 0.39	$ 0.78
25	CC0805KKX5R6BB105	4	$ 0.14	$ 0.56
26	CC0805MRX5R6BB106	2	$ 0.18	$ 0.36
27	0805ZC103KAT2A	2	$ 0.23	$ 0.46
28	CC0805KRX7R9BB102	2	$ 0.10	$ 0.20
29	APT2012EC	2	$ 0.30	$ 0.60
30	APT2012MGC	2	$ 0.66	$ 1.32
31	APT2012QBC/D	2	$ 0.57	$ 1.14
32	SC0918	2	$ 6.00	$ 12.00
33	TL2233OA	130	$ 0.37	$ 47.92
34	PH1-20-UA	4	$ 0.32	$ 1.28
35	61201421621	2	$ 1.25	$ 2.50
36	MCP23S09-E/SO	4	$ 1.63	$ 6.52
37	1729186	4	$ 5.29	$ 21.16
38	1725656	6	$ 2.00	$ 12.00
39	BA033CC0FP-E2	2	$ 1.41	$ 2.82
40	C0805C104M5RACAUTO	6	$ 0.14	$ 0.84
50	MCU Board Manufacturing	3	$33	$ 99.00

Figure 25 Bill of Materials Part 2