University of Puerto Rico
Mayagüez Campus
Mayagüez, Puerto Rico

Department of Electrical and Computer Engineering

Boardcaster: Final Report

by

Carlos N. Andreu-Martínez (Project Manager)
Richard B. Kaufman-López
Daniel A. González-Pérez
Francisco De La Cruz-Sánchez
Gabriel J. Pérez-Irizarry

For: Prof. Jose Fernando Vega, Prof. Wilson Rivera Gallego, and Prof. Kejie Lu
Course: ICOM 5047
Revised Date: December 17, 2011.

Table of Contents

List of Figures and Tables

Executive summary

The product presented is an electronic chess board that aims to solve the problem of chess enthusiasts not having a convenient way to watch chess matches. The electronic board is capable of detecting when magnetic pieces are moved over it in play and can transmit live chess matches over the Internet without the need for any drivers or platform specific software. Those interested in watching a live chess match only need a modern web browser and an Internet connection. To achieve this, a server connected to the Internet will be used as a middle-man. The electronic chess board will send the chess moves to the server, and the website will read from the server's database to graphically display the chess match on an Internet web browser. Our system also has the unique feature of illuminating valid moves for player when a piece is raised with lights located throughout each square on the board.

The team succeeded in developing an open-hardware, free and open source software which makes the product very customizable. It is also portable due to our battery powered design, wireless, real-time, easy to use and low cost compared to competing products.

The project expenditure totals at $672.31 as of the last day of development. This total includes a cost of $570.31 in parts required and $102.00 for the cost of shipping those parts to our facility. Notably we had to buy 64 LEDs at $0.19 each, 64 magnetic sensors at $1.91 each, the battery at $47.36, the micro controller unit (Arduino Mega) at $58.95 and the Wifi Shield at $89.95. The expenditure includes costs related to the production and shipment of the PCB as well and additional costs related to support and assembly of the two prototypes developed for the product.

Current and potential customers for Boardcaster include chess tournament organizers, any chess player from professionals to very beginners and due to the open hardware/software nature of the system, the team expects to attract people interested in expanding the feature set of the electronic board, the firmware and the server side software. The targeted market is very large because chess is one of the world's most popular games *[17]*.

Introduction

Chess tournaments are rarely *broadcasted*, if a chess enthusiast wants to watch a match, he or she must be at the event. This is disappointing for many; being at the event might not be a possible option due to travel costs or lack of time, among many other issues. As for the tournament organizer, not broadcasting the games results in less audience and less outreach. Boardcaster aims to solve that problem by offering chess enthusiast a convenient way to watch chess matches without leaving the comfort of their home and gives tournament organizers the possibility to reach more people.

The project is an electronic chessboard that is able to follow a chess match and “broadcasts” this match to a website. The system works by detecting chess pieces over each square on the chessboard. Each square has a sensor that detects whether a piece was picked up or placed on it; that way each move can be detected. Each move is then sent to a web server that accumulates these in a database in the server and shows them in a website so that anyone who wants to follow the game can graphically see it on this website. Additionally, each square also has an LED which can show the user the possible moves that they can make with a piece that they picked up.

Boardcaster has evolved from a design on paper to a real system. Its two main objectives have been met: a portable electronic chess board capable of broadcasting an ongoing chess match and a website where anyone can watch aforementioned games in real time. The products main specifications were successfully part of the last prototype: portability, wireless communication, easy to use, low cost and real-time. Details regarding those specifications and our objectives are within Design Criteria and Results and Discussion.

During the design, development and testing phases of the project the team gained valuable knowledge from a wide variety of sources. Notably Ruby on Rails Tutorial by Michael Hartl [22] helped with the website development, other resources include JQuery API page [23] and an online Git Tutorial [24]. Wikipedia [16] helped with general information about chess and how to represent the board using code. Research for the chess engine was based on engines that use bitboards [25]. However, the chess engine itself is based upon the original design by Ashwin Phatak [26]. Math regarding manipulation of bitboards in order to the generate possible moves for the chess pieces is based on the "Sliding Piece Attack" [27] and "Classical" [28] approaches. For the hardware various data sheets were used: Multiplexer [29], Decoder [30], Shift Register [31], Arduino [32], WiServer [33] and Hall Effect Sensor [34]. For the PCB creation[35] an Eagle to PCB tutorial [18] was very helpful.

This report will discuss the final status of this project. It is divided in six main parts: design criteria and specification, methods and approach to the solution, market overview, results and discussion, budget analysis, and conclusions and future work. It’s worth noting that references will be at the end in the bibliographic references section and the appendices will contain additional content referenced in the report. In the design criteria and specifications we describe the system and design details, tools used, constraints/limitations and minimum requirements for the system to run properly. Next we discuss methods and approach to the solution presenting an account of the activities in the projects, how the specs were tested and validated, summary of the testing and integration process, task divisions and details regarding the schedule. In the market overview portion of the report we present we identify potential users and compare our

system with our competition. For the results and discussion phase of the report we tackle technical results and ethical, legal, environmental and social aspects of the project. The final budget revision will present changes in the part list and changes in building costs from the last progress report. Finally, we conclude with the discussion of future work.

Design Criteria and Specifications

System specifications

This project is divided in two parts: an electronic chess board and a website.

The electronic chess board has sensors that detect chess pieces. Once a move is made, it records and sends that move to a server on the Internet. This is done using wireless communication.

The website graphically displays the moves made on the chess board. This will be watched by a web audience.

The electronic chess board features:

- Recognizes chess pieces on the chess board.
- Knows chess rules
- A LED will blink rapidly on the square when a piece has landed after an illegal move.
- Recognize a chess piece that has been lifted from the chess board.
- Lights up LEDs that will show all legal positions on the board for a piece that has been lifted.
- Recognizes a chess piece that has been placed on the chess board (Completion of a move.)
- Generates a FEN after each move, and stores it on memory.
- Sends the generated FEN to a server on the Internet.
- Connects to the internet wirelessly, using WiFi.
- Has a rechargeable lithium battery that provides 2200mAh.
- Portable
- Open-hardware

The server/website features:

- Has a database system.
- Has a web framework.
- Has a system of users and profiles.
- Receives data sent by the chess board.
- Shows, graphically, a chess board that represents the current status of a chess match.
- Updates, in real-time, the graphic chess board shown on the website.
- Has a search system for games.
- Allows replays of played games.
- Has a comments section on each game.
- Has votes/"Like" for every game
- Integrates with social networks like Facebook and Twitter.

Limitations

- No more than one Boardcaster chess board can broadcast a chess match to the server. (Maximum number of Boardcaster connections = 1). This was a design decision take consciously to lower the complexity of the website for the prototype.
- The chess board does not operate properly for fast chess play.
- The chess board requires careful placement of chess pieces.

Design Criteria

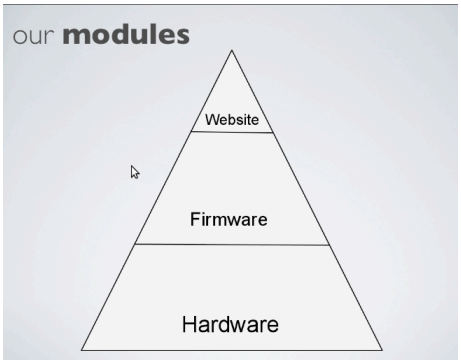


Figure 1: Boardcaster Sytem Modules

The team designed our system by dividing it into three modules; hardware, firmware and website. The design was built around the following design criteria:

Hardware

- Wireless - The boardcaster is wireless so that it can be portable. That means it has to communicate wirelessly and it needs to have a powerful enough battery.
- Portable - The boardcaster is portable by using Wi-Fi wireless communication and a Lithium Ion battery for power.
- Low power - Since the boardcaster is running on batteries it needs low-power compsumption so that the batteries can last for enough operation time.
- Affordable - In order to successfully compete in the market the team needs its product to be affordable. So they used low cost components and did a cheap to build PCB design.

Firmware

- Minimalistic and easy to understand - The team built a simple and clean implementation that is both efficient and easy to read. Also, the team needed the code to be easy to read for higher quality and for easier participation and collaboration.
- Real time - Since the firmware interacts directly with the user the firmware thas to be real time in order to give the users fast enough feedback.
- Low memory - In order to be affordable and portable the teams needs a memory efficient program so that the firmware can run well on microprocessors with very low specifications.

Website

- Look nice - The website has an attractive and modern interface for users. This is in order to attract more users.
- Easy to use - The website is easy to use so that it has a low entry level barrier.
- Watch live matches - Since it needs to be able to provide live matches the website needs to be fast and responsive.
- Leave comments - Users have the ability to leave comments to provide feedback to other chess players.
- Like games - For user interaction and community feedback the users are able to “like” games.
- User accounts - For user interaction and record keeping for users.
- Real time - Animations are carried out client side with Javascript and AJAX calls for automatic and fast website refresh.

Tools

PCB and Schematics

- Eagle CAD - PCB development and design software. [18]
- PSPICE - Schematic design and development software.
- Isopro - PCB printing software, used for PCB printing attempts at the University
- Quickcam - Gerber file exporter, used for compatibility with file formats

Hardware

- Multimeter - It is needed to debug hardware
- Oscilloscope - It is needed to debug hardware

Firmware development

- Arduino IDE - Interface to the Arduino compiler
- Microsoft Visio - Flowchart development software for firmware flowchart creation.
- Git - Version control system for software. [19]

Website development

- Ruby on Rails - Web framework for the website
- RSpec - Ruby testing framework used for website code testing [20]
- Gomockingbird - UI mock-up builder
- OmniGraffle - Flowchart development software for website design documents.
- Git - Version control system for software. [19]

Minimum hardware requirements

- Arduino Mega - Micro controller
- WiShield v2.0 - WiFi communication module
- Boardcaster PCB rev C - LED and sensor controller circuit
- WiShield compatible wireless router - Wi-Fi Internet provider for boards

- 64 boardcaster sensor/led squares - Sensor and display hardware
- 32 boardcaster magnetic chess pieces - For user interaction
- boardcaster chess shell - For user interaction
- 3.3 V 2200mAh Lithium Ion Battery - For board power

Minimum firmware requirements

- Avr-GCC 4.5.3 - For code compilation
- Arduino 22 - For project builds
- WiShield library 1.2.0 - For TCP/IP networking functionality for the firmware.
- DataFlash library 1.0.0 - A WiShield library dependency.
- Boardcaster firmware v 0.1 - For boardcaster overall operation.

Minimum website requirements

- Ruby >=1.9.0 - Programming language
- Ruby on Rails >=3.1.0 - Web framework
- Thin >= 1.3.1 - Web server
- PostgreSQL >= 9.1.2 - Data base server
- JQuery >=1.7.0 - Javascript framework
- Linux, Mac or Windows - OS

Minimum server requirements

Running on Ubuntu 11.10 server [21]

- CPU / Ram**:** CPU: 300MHz
- RAM: 128MB
- HDD Space: 1.1GB

Methods and approach to the solution

From the beginning, Boardcaster was divided in three parts: website, firmware, hardware. Team members were assigned to work in one or two of these three areas, each assignment done according to the abilities and knowledge of each member as described in Table 1. In this section we will be discussing the structure, organization, and the planning process that helped us produce the first working prototype of Boardcaster.

Team Member	Relevant Skills	Tasks
Carlos Andreu	Experience with Linux, web servers (Apache), web development (PHP, JavaScript, HTML, CSS) and databases (MySQL), team leader in previous projects. Relevant projects 'Yet Another Professor Rating' (yapr.org) and enterar.me.	Web development and project management
Richard B.	Wrote driver to interface LCD display with a Arduino Mega. Has	Firmware/Chess Engine design

Kaufman-López	experience with the C programming language. Has worked with wood and power tools.	and development. Board construction and testing.
Francisco De La Cruz	Has worked with many types of sensors including thermometers, sonars and GPS modules. Has also built interfaces for such sensors.	Hardware interfacing tasks and software for sensor interfacing.
Gabriel J. Perez Irizarry	Experience with Arduino, LED control and the game of Chess. Has written software for Arduino in C and assembly including additions to large existing code bases.	Hardware and software for the LED array.
Daniel Gonzalez	Has worked with and interfaced gyroscope sensors and has experience with different MCUs, has written software for Arduino in C and assembly	Hardware testing, PCB design and construction, board construction and connection.

Table 1. Each task was assigned to a member depending on his previous experience. This table describes this work division.

Website

Carlos Andreu was responsible for the design and development of the website. The first step in designing the website was to start painting a blank canvas with colorful specifications that will come to life at a later stage. It was key that the website had a user system and session management were new end-users will be able to register, login, logout, edit their profile and view other user's profiles. Sharing feedback was also a key component, end-users should be able to leave and read comments corresponding to a specific game, they should also be able to leave feedback via a "Like" and "Unlike" button. Since social media integration is very popular in the current market place the website must incorporate buttons for Facebook, Twitter and Google+ so people may share interesting games. The next step was to do a mock-up using GoMockingBird.com of the site's layout, focusing on a user-friendly interface guided by simple design principles (navigation in one place, clear and concise feedback messages, etc.). Next using Omnigraffle sequence diagrams were developed to illustrate how components interact (i.e. User visits login, session controller displays login form from login view, authentication in the model determines if the login is valid and the session controller creates a new session if credentials were successfully validated.). Use case diagrams gave our lead website developer a bird's eye view of the functionality certain types of users (registered and not registered) were able to access. UML Class Diagrams went into more detail. ER-Diagrams and the DB Schema detailed what type of data the website will be storing. Next step was to chose technologies to build the website. Ruby on Rails was the framework used, powering many popular sites like Twitter.com and Hulu.com after weighing the benefits and disadvantages. Testing was either done by manually going over the testing document or via automatic testing coded using RSpec Testing Framework.

Firmware

The firmware has four main functions: scanning the chess board and detecting the pieces; turning on and off the LEDs; making sure all the rules of chess are being respected and generating all possibles moves, on demand, for any chess piece; sending FEN strings with the current status of the board to the website.

Piece Scanner

Using decoders and multiplexers, the piece scanner polls reed switches and detects all the pieces on the board. It is constantly polling every square on the chess board and, when it detects a change, it passes that information to the chess engine.

Team member Francisco De La Cruz proposed the original design for the piece scanner. Since then, it never changed. De La Cruz also was in charge of the development of the piece scanner. Nor surprisingly, he was also in charge of the hardware side of the piece scanner, but more details of the hardware will be shared later in this section.

The piece scanner was individually tested using four reed switches. The code drove the multiplexers and decoders. We successfully were able to detect changes: a magnet was lifted, a change was detected, the magnet placed, and the change was detected again. These changes were reported and read on the Serial Communications window.

LED Driver

Similar to the piece scanner, the LED driver uses shift registers to control the LEDs. Gabriel Perez designed the code and hardware related to the LEDs; it never changed. These were individually tested in a similar fashion to the tests for the piece scanner. Four LEDs were being controlled by the LED driver code using shift registers. Results were verified visually: is the LED on or off?

Chess Engine

The chess engine was designed by Richard Kaufman. The design for the chess engine changed in many occasions. Although all of them were functional designs, all but the final design consumed too much memory. In the section “Results and Discussion” of this document more details, including tests and results are discussed.

Wireless Communication

Wireless communication was achieved with a WiFi Shield. The design for the code is very simple; receive a FEN from the chess engine and send it to the website. Gabriel Perez worked with this task. The library supplied by the vendor of the WiFi Shield is well documented and provides and excellent API. In the section “Results and Discussion” of this document more details, including tests and results, are discussed.

Hardware

After the design phase, the hardware component of the project was divided into three main subsystems: the sensor array, the LED array and handling all the connections from the sensors to the microcontroller. It was decided that one member each would handle each of these separately, test them and then integrate the three of them. Gabriel was tasked with the LED array, Francisco with the sensor array, and Daniel with finding a way of handling the connections.

For the sensor array, the first task was to decide which magnets to use and which reed sensors so that the magnets were strong enough to activate the sensor, yet not strong enough that the magnets would stick if the pieces were put side by side. For this, several test magnets and sensors were ordered to test the different strengths of the magnets, and after these tests one combination of magnet and reed switches were selected as the one to be used for the project. Afterwards, a small circuit consisting of the decoder and two multiplexers and a couple switches were used to write and debug the firmware, and after testing everything this hardware part was done.

As for the LEDs, the first task was to select the drivers for them. After looking for several options, the group settled on using four 16-bit shift registers, where each bit held on the register is used to signal the LED on or off. After getting the part, a small circuit with two registers and a couple of LEDs was used to write and debug the firmware, as explained before. When the code was debugged, this component was considered finished.

The connections for all these sensors and LEDs were so many that the group looked for a way to manage them. It was decided to design and print a PCB that would have the components soldered to it, and it would connect to the sensors using ribbon cables. The design was done and tested using Eagle. First the group tried to print the PCB in one of the labs in the university, but they did not have the necessary milling and drilling bits, and the printing could not be done, as the PCB required some fine prints, so the PCB was ordered from a company, and when it arrived, all the components were soldered to the board. It was tested by connecting it and setting a couple of the reed sensors and LEDs in a board and connecting them to the PCB. After testing it worked, the hardware component of the project was done, and as soon as the code was too, integration began.

All the components and their individual testing went according to schedule, except for the PCB. What happened with the PCB, as explained before, was that Francisco, Daniel and Gabriel tried to print it on a campus' lab, but found out half way through getting the machine to work that the necessary drill and mill bits were not available, and buying them would cost way more than ordering the PCB from a company, so it was decided to order it. This was a delay, but did not affect the critical path of the schedule since this part was required only when the integration was done, and everything else could proceed regardless.

Chess Board Box

The chess board box task was assigned to Richard Kaufman. The initial design and plan consisted in using a CNC wood cutting machine to cut the pieces that made up the chess board box. However, this plan was abandoned for two reasons. Production of the chess board box would have been expensive. Furthermore, we already had a chess board box we could recycle, and that was similar to our design. This saved money and time. Had we proceeded with the original plan, we would have had to redo the 3D model of the box in AutoCad; previously it was designed using Google's SketchUp. Poor experience with this software would have delayed the production of the first prototype. Testing of the board was done by the whole team.

Tests

For more information regarding tests, including tests for system specs, please refer to the section "Results and Discussion" and Appendix C: Testing and Specification Compliance Documents.

Integration

When integrating, all the hardware worked well, except for a few faulty sensors that had to be replaced. The website also worked correctly. As for the firmware, everything was working correctly, but when everything was put together, it was seen that the microcontroller would randomly reset itself, and the group determined it was a memory problem. This is further described in the results and discussion section. After noticing the problem, the group divided itself to find alternatives: Gabriel and Francisco would test a new library to see if it was feasible to replace it with that one, Richard and Carlos tried to fix the old code, while Daniel kept on connecting and gluing everything to the board, with help from the others whenever they could. This memory problem caused the final delay, but after fixing it everything was done and worked flawlessly. Although this was a delay, the contingency plan was to start integrating everything a week earlier than schedule to mitigate the integration problems that were expected and did occur.

Market overview

There are a wide variety of potential customers available for this product. The most obvious customer for this product would be chess tournament organizers, because our system makes it very easy to broadcast to large audiences. Another substantial customer base would be advanced and professional chess players, they could benefit from *"boardcasting"* their games to gain more fame and recognition. Also, beginner chess players will appreciate having visual feedback of valid moves when they raise a piece, and every player may improve their game by reading comments from the chess community through the game's website. Finally as an open-source hardware and software project this product would be of great interest to hackers, this can also be yet another customer base. The hacker community could show interest in this product as a platform for creating new games that the team hasn't even considered yet or that they do not have the resources to develop. The market we're targeting is composed of many people, since chess is one on the world's most popular games, played by millions of people worldwide at home, in clubs, online, by correspondence, and in tournaments *[17].*

Market Overview Table

Name	Platform	FOSS+ OSS Hardware	Price	Wireless	AI	Automatic move detection	Records games	Training	Online play	Broadcastin
Boardcaster	Platform independent (web based)	Yes	$425.04	Yes; Wi-Fi	No	Yes	Yes; through website	Yes; Simple training through LEDs and comments through webiste	No	Yes; through the web
DGT	Microsoft	No	€649.00	Yes;	Yes; but	Yes	Yes; but	No	Yes	Yes; but

	Windows		(~$890.00)	Bluetooth	only when connected to a PC.		has space for only 500 moves internally.			requires special software to watch
Professional Tournament Manager Hub	Microsoft Windows	No	$489	Yes; Bluetooth	No	No; requieres manual entry	Yes	No	No	Yes; through the web
Shacom	Microsoft Windows	No	$480	No	Yes; but only when connected to a PC.	Yes	Yes	Yes; this is a special focus of this product	Yes	Yes; but requires specialized additional equipment

Table 2: Market Overview Table

Other competitors

Potential competition could come from the current providers of electronic chess boards and chess tournament equipment such as the ones previously mentioned. Since the team developed an open-source software and hardware product competition could spawn in the form of spin-offs based on our design and code. Fortunately, by making use of copy-left licenses the team is able to take advantage of advances made by other developers. Copy-left will protect the end-users and developers from the product becoming proprietary but it won't protect the developers against parasite competitors that only manufacture the product but don't contribute back to its development. The team can only hope that the product's quality and customer loyalty by themselves shield the product from parasitic competitors [8]. This model has worked in the past for other open-source-hardware projects such as the Arduino and Arducopter projects.

Results and Discussion

The team succeeded in creating a product that complies and solved the following problems:

Portability

The product includes a rechargeable battery to power the chess board. The owner of the chess board will also be able to take it and move it around. Battery life will depend on usage, but during internal tests the battery lasted more than 60 minutes. According to research done by the team, the average chess game lasts between 10 and 60 minutes *[16]*. Hence, it is good enough for most typical use cases (a normal game). However, owners may extend their battery life by turning off the LEDs in charge of providing visual feedback.

Wireless Communication

Using wireless communications, the need for a cable connecting the board with the computer is non-existent. In fact, it removes the need for a computer altogether. The only requirement would

be a router providing Internet access. Boardcaster will communicate to the Internet and update the server as moves are played. The maximum FEN (Forsyth-Edwards Notation) size is about 85 bytes. On top of this we must consider the average HTTP POST header of about 200 bytes for a total of 285 bytes per move for the largest FEN.

Bandwidth is usually not an issue, since for example: Choice Cable in Mayaguez provides customers with 12 Megabits per second for 30 dollars *[17]*. The time it takes to transmit these moves is only conditional on network latencies as processing time is negligible.

Easy to use

There is no need to install drivers or support applications for that matter. Boardcaster owners just need to turn it on in a place with wireless Internet access, register on the website and start a game via the easy-to-use web interface.

Low cost

Although the project total part cost listed for the project is of $570.31 the actual part cost for a production demo will be lower. The provided sum of $570.31 includes the costs of producing two prototypes as well as one-time expenditures related to the production of both prototypes. If taken to market the team feels confident there is an ample gap between the Boardcaster and the nearest competitor. For example, Digital Game Technologies produces a similar product that retails at €509.00 for a non-wireless chess board [18]. Even after adding a markup to the price, Boardcaster still remains an economical and competitive product. After the lessons learnt during the production of the two prototypes improvements, more efficiency and mass-production will bring the price down.

Real-Time

As soon as a player moves a chess piece, the move is published on a website. Everyone watching will see this move within a 10 second time frame, assuming good network connectivity and normal server load.

Technical Problems

Chess Engine

Most chess engines are designed to be fast. In order to be fast, they must have instant access to hundreds, if not thousands, of possible moves; moves must be pre-computed and stored somewhere accessible to the chess engine. In computers this information is pre-generated, or perhaps read from disk and stored in random access memory before the chess engine even starts playing. But, how do you approach this problem, namely having instant access to thousands of moves given a state of the chess board? Well, you don't. However, to reach that conclusion it took ACM5PT various cycles of development, testing, and debugging. In this section we will discuss the development, testing, and debugging approaches that helped us achieve our goal of having a chess engine inside the Arduino Mega 2560.

The first version of the chess engine consumed more than 256KB of RAM; an Arduino 2560 only has 8KB of RAM. Still, this was not immediately an issue. Most of the development for the engine was done early in the semester, the code was not uploaded to the Arduino. Instead, it

was run and tested on a computer. For the first month of development, memory was not an issue. Furthermore, the chess engine behaved as expected: it would always keep record of the game status; would correctly print all the possible moves for a given chess piece, including special moves such as castling and preventing the king from moving to a “check” square; pawn promotion worked correctly although always automatically promoting to a queen. All in all, it already knew all the chess rules.

The next stage consisted on uploading the engine’s code to the Arduino. While it did upload, it did not execute properly. At this stage only the code for the chess engine was in the Arduino.

The first problem that hinted towards lack of memory was that the Arduino would keep resetting itself. It was not until weeks later that we found out why: an *assert(...)* would fail. In a computer an assert would terminate the process. On a Arduino, the process is restarted. In hindsight, the *assert* failed because the variables that were being tested that were never properly set. With a full memory stack it is impossible to determine the values of any variable. Furthermore, multiple executions of the same program would give different values to some variables. Again, this explains why the process would reset itself at random places during execution.

Diagnosing a problem such as memory exhaustion was possible due to the team’s previous experience with development on other, similar, platforms. Manual calculation of memory consumption –counting variables types and summing byte usage, e.g. an integer uses 4 bytes, a *bitboard* 8 bytes– made it clear that the code was exceeding the RAM available to the device upon execution.

The next version of the code had improvements in memory consumption. Originally many moves were pre-calculated and stored in *bitboard[64][64]* arrays, each being 32KB big. Removal of these lowered memory consumption to an estimated maximum of around 9KB. Again, when uploading and running this program it would behave incorrectly.

Further memory improvements of the same fashion –removing unnecessary arrays– brought memory consumption to a mere 5KB. While this is still a huge amount for a microcontroller, it was low enough to allow the chess engine to run correctly.

All in all the development team had to sacrifice speed for low memory use. After all, speed improvements at the level provided by the code in its original form was not of importance to the product. Had it been an A.I. vs A.I. environment, where even a millisecond matters, the situation would have been different. However, a less than one second delay for move calculation is acceptable; it does not downgrade gameplay.

WiFi Shield

After the chess engine was inside the Arduino along with the LED driving code and the piece scanner, only integration with the WiFi Shield was missing.

The shield had been tested before. In fact, it had been presented working properly and communicating with a test website. Somewhat unexpected were major issues related to our old

friend, a full memory stack. This time we had tools to calculate how much memory the code was consuming, and how much memory the process was lacking. Results showed that the Arduino was running short by 2KB of RAM.

Improving the memory economy this time seemed like a daunting task; memory consumption had already been chopped down from the 250KB of RAM usage the chess engine required. Where else could improvements be made? The WiFi Shield library was not an option. Besides this being the official library and proven to work, this code base was mostly unknown to the team. The LED driving code was plain simple, and so was the piece scanner. No space to improve there. Bravely, the team went back to the chess engine determined to make it even more efficient memory-wise. Eventually the group struck at gold and found more code segments where savings of about 4KB were made. In the end the RAM consumption by the engine hovered around 2KB.

As a direct result, the WiFi Shield, along with everything else, executed correctly.

Social impact

Boardcaster brings people together. Users are able to easily watch chess matches, leave comments on them, vote for the games they like, and share it with their friends. Additionally, it also connects chess tournaments, clubs, and organizations with the rest of the world. In this matter it levels the playing field. Long gone will be the need for big sponsors such as ESPN to be able to transmit a chess match to audiences with cable TV. Now anyone with a Boardcaster chess board can broadcast games to the world. On the other hand, the group feels it has significantly revolutionized the requirements to watch a chess match: only a web browser and Internet connectivity are needed. This is important today because of the wide availability of devices having both: tablet PCs, smart mobile phones, and computers. This is a big improvement over the competition's requirements: a computer running Microsoft's Windows, proprietary software that had to be installed, and configuring connections to a server. Money and human resources are no longer the barrier to reach those outside the building hosting the chess event.

Environmental issues

All the selected electronic components that are part of the chess board are compliant with the RoHS (Restriction of Hazardous Substances Directive) standard. This means that the electronic components in the chess board do not contain any of the following hazardous materials: Lead, Mercury, Cadmium, Hexavalent chromium, Polybrominated biphenyls, or Polybrominated diphenyl ether. These six materials are hazardous to humans and the environment. It reduces damage to people in third-world countries, where most electronic trash ends. RoHS has also provided immediate health benefits to workers in the electronics industry.

Furthermore, for the first prototype recycled chess board boxes were used. However, when full production starts, the team wants to ensure that the wood used to prepare these boxes does not come from trees that are on the verge of extinction or from over-exploited forests.

ACM5PT is fully committed to being friendly to the environment and to everyone that inhabits it.

Legal considerations

ACM5PT must be careful not violate any already existing patents. Although research was done on solutions similar to Boardcaster, none were found. Further search would require more time, and perhaps lawyers. In the case of patent violations, Boardcaster's design could be jeopardized; the group could be sued by a patent holder, which is why research for any similar products will be done exhaustively before production to market starts.

Ethical considerations

Reaffirming ACM5PT's commitment with humanity the team also believes in fair pay and humane treatment of workers. Working environment and employee treatment standards will be taken into consideration when deciding which factory will finally produce the Boardcaster.

However, there is a scenario where ACM5PT might not have control over these issues. Since the product is open-source and open-hardware, anyone in the world might also produce copies of Boardcaster chess boards. If an individual or company with poor ethical standards decides to reproduce the board, they might outsource the work to countries where workers are exploited.

In light of the group's commitment to being open about the software and hardware it produces, there is no simple way of stopping these individuals. Instead the team can only promote the set of beliefs that drove to a successful product completion and hope that others follow.

Budget Analysis

As of December 17, 2011 project non-labor expenditure totals at $672.31. This total includes a cost of $570.31 in parts required for the purchase and shipment of components and services related to the PCB construction. This cost also includes the total required count of multiplexers, decoders and shift registers including one spare of each. Additionally $62.45 that went towards the cost of the Wifi-Shield. A costs saving of $27.5. The remaining amount accounts for the cost of the PCB drill bits, the copper clad plates, glue, additional magnets and other prototype construction related components.

Compared to the team's estimated parts cost of $480.01 the sum implies budget overrun of $192.3. Fortunately, the team prepared for an overhead of up to 180%. Taking into consideration this margin the project's non-labor expenditures represent an overhead consumption of 25%.

From the project start date of September 17, 2011 until the final work date of November 13, 2011 58 project work days elapsed. Labor and benefit costs were estimated at $20503 for the length of the project which was estimated at 58 days.

Actual labor and benefit costs were calculated at $21,176.18. This change is solely related to the parts cost increases attributed to both prototypes produced.

Three main changes were made to prototype construction and part choices. The proposal projected the construction of one prototype by hand and utilizing wooden and metal materials. The parts cost for this process was estimated at $24.35. Unfortunately, due to time constraints the team had to resort building the board in house.

To facilitate the placement of 128+ components across the board the team opted for a PCB (Printed Circuit Board) design. Time and monetary costs induced by the design and manufacturing process incurred in additional costs and a tighter schedule. Finally, the team opted for reed switches over Hall-effect sensors for piece detection due to the simplicity, low power consumption and relatively low prices. By using reed switches the team saved 20 cents per unit.

The Table 2 below shows a Budget Analysis Summary. This table summarizes the final budgetary state of the project. *Final Parts Cost* in this summary include parts cost and shipment as well as the cost supplementary tools and materials related to cost of building both prototypes.

Final Parts Cost	$672.31
Projected Cost	$480.01
Absolute Cost Maximum	$768.02
Current Budget Status	-$192.30
Overhead Cosumption	25.04%

Table 3: Budget Analysis Summary

In the appendix you will find additional tables outlining the budget analysis costs utilized for the budget summary derivation. A Parts list, Costs per store as well as an Employment costs table is also provided.

Conclusions and Future Work

Although a few delays were encountered, the project was a huge success. As described in this document, it fulfilled all the requirements specified, and worked as expected. The approach taken by the group is considered to be a good one, since even with the delays encountered, countermeasures could be taken to deal with them and not affect the rest of the project; the only major delay was with the integration, but it was still dealt with and every planned task was completed.

This product fulfills the group's expectations, and it is expected that if it were to be produced, there would be a great market for it, as explained throughout this document. The biggest challenge aside from technical ones would be to make sure Boardcaster does not infringe on any patents, as lawsuits could severely damage its production. Aside from this, there are no other major drawbacks: the price is still lower than that of competing products, even when adding the cost for labor and development. Additionally it is environmentally friendly and has a

great potential for a positive social impact by connecting people together to play and watch chess matches and even learn from each other.

The first step for improving Boardcaster is to find a more effective way of assembling the board so that the sensors and LEDs will stay in place and proper contact can be made with the magnets in the pieces. Another quick improvement would be to handle more errors from the user, such as telling them they are in the wrong square and tell them to go back.

The group has quite a few improvements in mind for making Boardcaster even better. First of all, the website should be able to track more than one board at a time, which was not done due to time constraints. It would also be good to be able to play against artificial intelligence (AI) programs that play chess, which would allow for a whole new way to play. This would also give the board a new potential market, as AI researchers would have a platform where they can test their programs against real players, and would allow people to play by themselves and save these matches on the website for others to see and maybe even to teach others how to play.

Bibliographic References

[1] DGT, "DGT Bluetooth Wireless e-Board." [Online]. Available: http://digitalgametechnology.com/site/index.php/Electronic-Boards/dgt-bluetooth-wireless-e-board.html. [Accessed: 07-Sep-2011].

[2] "arducopter." [Online]. Available: http://code.google.com/p/arducopter/. [Accessed: 07-Sep-2011].

[3]Francisco De La Cruz Sanchez, Daniel A. Gonzalez Perez, Richard B. Kaufman Lopez, Carlos Andreu Martinez, and Omar Ferrer, "Reporte Final Arducopter."

[4] "Occupational Employment Statistics," 04-May-2009. [Online]. Available: http://stats.bls.gov/oes/2008/may/oes151021.htm. [Accessed: 07-Sep-2011].

[5] "Professional Tournament Manager Hub." [Online]. Available: http://monroi.com/products/professional-chess-tournament-manager.html. [Accessed: 07-Sep-2011].

[6] "Human Resources Benefits." [Online]. Available: http://www.umdnj.edu/hrweb/benefits/index.htm. [Accessed: 07-Sep-2011].

[7] Shahcom Company, "Electronic chess board 'Shahcom'."[Online]. Available: http://www.ruschess.com/Store/Access/board.html. [Accessed: 07-Sep-2011].

[8] Clive Thompson, "Build It. Share It. Profit. Can Open Source Hardware Work?" [Online]. Available:http://www.wired.com/techbiz/startups/magazine/16-11/ff_openmanufacturing?currentPage=a. [Accessed: 07-Sep-2011].

[9] "USB Roll-Up Chess Game," *amazon.co.uk*. [Online]. Available: http://www.amazon.co.uk/5051494010001-USB-Roll-Up-Chess-Game/dp/B000WX1FA4. [Accessed: 07-Sep-2011].

[10] "Architectural and Engineering Managers." [Online]. Available: http://www.bls.gov/oes/current/oes119041.htm. [Accessed: 13-Sep-2011].

[11]"Computer Hardware Engineers." [Online]. Available: http://www.bls.gov/oes/current/oes172061.htm. [Accessed: 13-Sep-2011].

[12] "Calculating Overhead Percentages." [Online]. Available: http://www.missouribusiness.net/sbtdc/docs/calc_overhead_percentage.asp. [Accessed: 13-Sep-2011].

[13] "Bureau of Labor Statistics Data." [Online]. Available: http://data.bls.gov/timeseries/CUUR0000SA0?output_view=pct_12mths. [Accessed: 13-Sep-2011].

[14] "Canadian Industry Statistics - SME BenchmarkingCanadian Economy (NAICS 11-91)." [Online]. Available: http://www.ic.gc.ca/eic/site/cis-sic.nsf/eng/h_00032.html. [Accessed: 13-Sep-2011].

[15] Honeywell, *Hall-Effect Sensors*. Honeywell, 2011.

[16] "Chess" [Online] Available: http://en.wikipedia.org/wiki/Chess

[17] "Choice Cable" [Online] Available: http://choicecable.com/

[18] "Turn your EAGLE schematic into a PCB" [Online] Available: http://www.instructables.com/id/Turn-your-EAGLE-schematic-into-a-PCB/

[19] "DGT e-Board" [Online] Available: http://www.newegg.com/Product/Product.aspx?Item=9SIA02200025T4

[20] "Sn introduction to RSpec - Part I" [Online] Available: http://blog.davidchelimsky.net/2007/05/14/an-introduction-to-rspec-part-i/

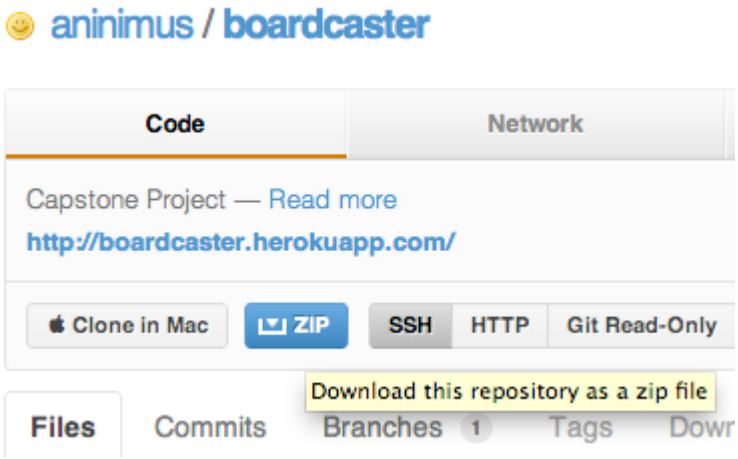
[21] “Installation System Requirements” [Online] Available:
https://help.ubuntu.com/community/Installation/SystemRequirements
[22] “Ruby on Rails Tutorial by Michael Hartl” [Online] http://ruby.railstutorial.org/
[23] “JQuery API” [Online] http://api.jquery.com/
[24] “Git Tutorial” [Online] Available: http://www.vogella.de/articles/Git/article.html
[25] “Bitboards” [Online] http://fiascochess.wordpress.com/2009/03/15/writing-a-bitboard-chess-engine/
[26] “Chess” [Online] https://github.com/ashwinphatak/chess
[27] “Sliding Piece Attacks” [Online] http://chessprogramming.wikispaces.com/Sliding+Piece+Attacks
[28] “Classical Approach” [Online] http://chessprogramming.wikispaces.com/Classical+Approach
[29] "Multiplexer Data Sheet" [Online] http://datasheets.maxim-ic.com/en/ds/MAX4581-MAX4583.pdf
[30] "Decoder Data Sheet" [Online] http://www.nxp.com/documents/data_sheet/74HC_HCT138.pdf
[31] "Shift Register" [Online] http://media.digikey.com/pdf/Data%20Sheets/Sharp%20PDFs/IR2D07.pdf
[32] "Arduino / Hardware" [Online] http://www.arduino.cc/en/Main/hardware
[33] "WiServer" [Online] http://asynclabs.com/wiki/index.php?title=AsyncLabsWiki
[34] “Hall Effect Sensor Data Sheet” [Online]
http://www.digikey.com/Web%20Export/Supplier%20Content/Meder_374/PDF/MEDER_Reed_Sensors_vs._Hall_Effect_Sensors.pdf
[35] “How to make a PCB” [Online] http://ece.uprm.edu/~s087918/Manuals/How%20to%20make%20a%20Pcb.pdf
[36] “Boardcaster Final Presentation” [Online] https://www.dropbox.com/s/3jqvur9a5xr34f3/Capstone-Final-Presentation.pdf

Appendices

Appendix A: Code Repository

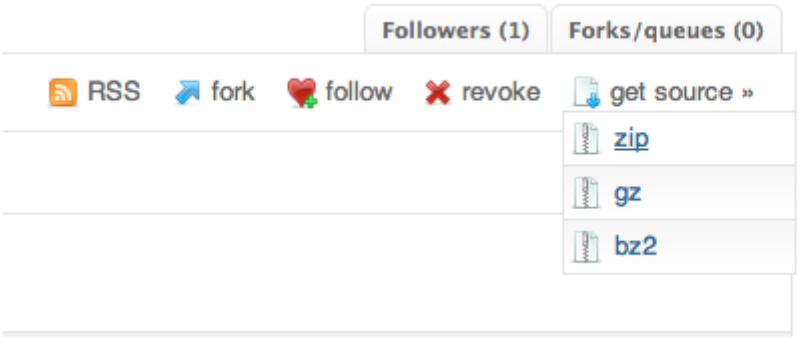
(Website, Firmware and Hardware Schematics)

All the fully documented code for the website is available online here:
https://github.com/aninimus/boardcaster



Appendix A1. To download the code simply click on the "Zip Button".

All the fully documented code for the firmware and the hardware schematics are available online here:
https://bitbucket.org/xerces/boardcaster/src



Appendix A2. To download the code simply click on the "zip" link under "get source".

Appendix B: Web Log (Blog)

Our blog with over 100 posts detailing our meetings, problems and milestones is located here:
http://blogs.uprm.edu/boardcaster/

Appendix C: Testing and Specification Compliance Documents

Module: Website

Metrics
1) Pass - Behavior is as described
2) Fail - Behavior is not as described

Sub-Module: Layout
- Should have navigation links for unregistered users:
 * Home
 * Create Game
 * View Games
 * Sign up
 * Login
- Should have navigation links for registered users:
 * Home
 * Create Game
 * View Games
 * Profile
 * Logout
- Footer should have links for:
 * About
 * Help
 * Contact
 * Back to top - should bring the page to the top
- Navigation bar should have a search field
- Layout should be consistent across all pages
CURRENT STATUS: All Pass

Sub-Module: Main
- Should be located at: /
- Title should be "Boardcaster | Home"
- "Home" should be highlighted in the menu bar
- Registered users should see
 * Register Now button (blue)
- Unregistered users should see
 * View Games button (green)
CURRENT STATUS: All Pass

Sub-Module: About
- Should be located at: /about
- Title should be: "Boardcaster | About"
CURRENT STATUS: All Pass

Sub-Module: Help
- Should be located at: /help
- Title should be: "Boardcaster | Help"
CURRENT STATUS: All Pass

Sub-Module: Contact
- Should be located at: /contact

- Title should be: "Boardcaster | Contact"
CURRENT STATUS: All Pass

Sub-Module: Create Game
- Should be located at /create
- Title should have "Boardcaster | Create a Game"
- "Create a Game" should be highlighted in the menu bar
- Unregistered User
 * Should redirect to the Login form
 * Should display "First login to access this page" (yellow)
- Registered User
 * Should redirect to current game if game is in progress, should display "Game in progress" (red)
 * When no game is in progress it should allow users to create a game by typing a title and selecting a black and white player
- Validation before allowing data in the database
 * Both users must exist in the database (black and white players)
 * Both users must be different (black and white players)
 * User ids should be numeric
 * Title must be unique, validation should be is case-insensitive (eg. "TItlE" is equal to "title")
 * Game must be flagged as live
 * live must be either "true" or "false"
- After a game has been created it should display "Game created!" (green)
CURRENT STATUS: All Pass

Sub-Module: Show Game
- Should be located at /games/[id] (eg. /game/20 for game with id=20)
- Title should have "Boardcaster | Game | [Game Title]"
- User must be registered to view this page, otherwise redirected to login page
- Players will be displayed
- Chessboard will be displayed
- Comments will be displayed
- User may like (show blue "Like Game" button) and unlike games (show green "Unlike Game" button)
- Validation before allowing changes to like and unlike games in the database:
 * User must exist
 * Game must exist
 * User may not like a game twice
- User may send comments
- Validation before allowing comments to be written to the database
 * User must exit
 * Game must exist
 * Maximum length must be 750 characters
 * Minimum length must be 1 character
 * If successfully posted it should display "Comment created." (green)
 * If it failed to be posted it should display "Invalid comment." (red)
CURRENT STATUS: All Pass

Sub-Module: View Games
- Should be located at /games
- Title should have "Boardcaster | View Games"

- User must be registered to view this page, otherwise redirected to login page
- Should list all games
- Clicking on games should redirect to to that game's show page
CURRENT STATUS: All Pass

Sub-Module: Profile
- Should be located at /users/[id] (eg. /users/20 for user with id=20)
- Title should have "Boardcaster | Profile | [username]"
- Should display user's name and email address
CURRENT STATUS: All Pass

Sub-Module: Sign up
- Located at /signup
- Title should have: "Boardcaster | Sign up"
- Navigation bar "Sign up" should be highlighted
- Unregistered users may click "Sign up" and fill out: username, email, password and password confirmation
- Validation before a user is written to the database:
 * Username maximum length must be 40
 * Username must be unique (case insensitive validation)
 * Email maximum length must be 255
 * Email must be valid (regex)
 * Email must be unique (case insensitive validation)
 * Password and password confirmation fields must be equal
 * Password must be within 6 and 40 characters
- Display form is invalid with error list when user enters invalid data (red)
CURRENT STATUS: All Pass

Sub-Module: Logout
- Located at /logout
- Should delete login cookie
- Should redirect back to main page
- Should display "Logged out!" (yellow)
CURRENT STATUS: All Pass

Sub-Module: Login
- Located at /login
- User may type his username, password and optionally check a Remember Me checkbox
- If the data entered was incorrect display "Email or password was invalid" (red)
- If the data was correct redirect back to the page the user was before beign redirected to the signup form and display "Logged in!" (green)
CURRENT STATUS: All Pass

Sub-Module: Moves
- POSTs to /moves/?move_data=[fen] should record a move to the database
- GETs to /moves/ should return the last move
- Validation before a move is written to the database
 * Move_data can't be blank
 * Move_data minimum length should be 28
 * Move_data maximum length should be 255
 * Move_data must be valid (using regex)

* Game_id must be numeric and it must exist
CURRENT STATUS: All Pass

Module: Hardware - Piece Detection

Metrics

1) Pass - Behavior is as described
2) Fail - Behavior is not as described

Mechanisms

Visual confirmation via testing equipment such as oscilloscopes and or multimeters and or serial debug output.

Description

The Piece Detector Demo consists of a decoder and two multiplexers synchronized to read an array of 6 reed switches. The demo shows simultaneously the correct select signal generation to enable both multiplexers by the decoder and the correct signal select by the multiplexers showing the desired reed switch state. Both multiplexer outputs are connected to a common bus line allowing for a reduction in pin usage and a simplification in circuit wiring and design.

The demo, and eventually the full piece detector module will make use of 6 signaling lines, power rails and a common bus line for a total of 7 data lines and the VCC and GND line. The demo operates at +5V but can also operate at +3.3V if so desired. Below we show an execution of the piece detector firmware.

```
Initializing Piece Array Scanner  [323]ms/scan..
Initialization Done.
[
1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
[
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
[
```

As seen above, initializing the board reed array consists of polling all switches and calculating the approximate array poll traversal time. After this, the software only shows board state changes. In the output above we show the result of placing a magnet on the first reed switch and removing it.

A 1, corresponding to a logic HIGH, is shown in position 0,0 of the array corresponds to placing the magnet above the reed, this state change is shown by printing the new bitboard status. Once the magnet is removed, the board status changes once more.

Test Cases and Their Results

Case 1

Reed sensor state read successfully by the Arduino board.

Test [PASS]

Develop firmware code to test switch state changes upon presence or absence of a magnet perpendicular to switch. This test result is implicitly visible in the demo.

Case 2

Decoder generates the appropriate signals to enable the corresponding multiplexer.

Test [PASS]

By connecting the decoder outputs to the multiplexer enables and ensuring the appropriate output for the multiplexer we ensure the multiplexer is being enabled correctly by the decoder. Also, by generating a steady and predictable signaling pattern and oscilloscope was used to confirm correct signal sequencing.

Case 3

Connect multiple multiplexers to the decoders and verify correct switching and selection.

Test [PASS]

Implementing a board sweep algorithm and device signaling routines allows us to ensure the correct wiring and operation of the devices. By manually testing a set of individual switches and printing intermediate status boards we can confirm correct enabling of the multiplexers and correct reed selection by the multiplexer.

Case 4

Ensure a full board **scan time** of under 10 seconds.

Test [PASS]

By setting a timer at the beginning of the piece detector scan algorithm and comparing the time at the exit of the routine we can get the average time to scan the board. **Results:** Under normal operation the average scan time of the whole board was measured at **324ms**. This allows the Boardcaster to hypothetically scan the board 30 times in a 10 second time slow.

Module: Hardware - LED Array

Metrics

1) Pass - Behavior is as described
2) Fail - Behavior is not as described

Mechanisms

Visual confirmation of the LEDs

Description

The LED Array Demo consists of two shift daisy-chained shift register with a few LED bulbs connected to each one. The demo shows simultaneously the correct serial signal generation to display data with the LEDs. The demo, and eventually the full piece detector module will make use of four latch and four XEN signaling lines, one clock and one serial data line for a total of 10

data lines and the VCC and GND line. The demo operates at +3.3V but can also operate at +5V if so desired.

Test Cases and Their Results

Case 1

Use a single shift register and shift out data to turn on LED.

Test [PASS]

Develop firmware code to generate clock, latch, xen and serial signals to control the shift register and connect LEDs to shift register. This test result is implicitly visible in the demo.

Case 2

Daisy-chain connect two shift registers and display data with multiple LEDs.

Test [PASS]

Two shift registers were daisy chained and data was shifted out for both. The result is that LEDs connected to either shift register can be controlled.

Module: Hardware - WiShield

Metrics

1) Pass - Behavior is as described
2) Fail - Behavior is not as described

Mechanisms

FEN data is received by the boardcaster website.

Description

The WiShield Demo consists of an Arduino that has the WiShield mounted. The demo shows that the WiShield can succesfully connect to a wireless network and can send FEN data to the boardcaster web through an HTTP POST.

Test Cases and Their Results

Case 1

Connect WiShield to wireless network.

Test [PASS]

The WiShield was configured to connect to a router and the WiShield connection indicator LED turned on.

Case 2

Send HTTP POST with FEN to boardcaster website.

Test [PASS]

A WiShield library function was used to execute the POST. The POST can be confirmed as successful by looking at the boardcaster website.

Module: Firmware

Metrics

1) Pass - Behavior is as described
2) Fail - Behavior is not as described

Description

The chess engine is the heart of the firmware. It has two main purposes: knowledge of all the rules of chess, and the calculation of the legal moves for a specific chess piece. In addition, the chess engine is also in charge of generating the FEN strings that will be sent to the Boardcaster.com web server.

Test Cases and Their Results

Case 1

To be able to do the aforementioned, it is necessary that the chess engine has knowledge of all the chess rules. Not even a minor bug is acceptable here. This case is, therefore, the most important of all.

Test [Pass]
Check correct evaluation of basic moves such as: a simple move, an attack, piece promotion, castling, and an *en passant* attack.

A simple move consist on lifting one piece from one square and placing it in a different square. If the move is valid, return true.

An attack move consist on lifting one piece from one square and placing it in a different square occupied by the opponent's piece. If the attack is valid, return true.

Promotion of a piece consist on a pawn reaching the final rank of the chess board (relative to that player's piece.) The pawn can then be exchange with any other piece, although usually a queen. If the promotion is valid, replace the piece with a queen.

Castling is a move that consists of moving the king two squares towards a rook on the player's first rank, then moving the rook onto the square over which the king crossed. Castling can only be done if the king has never moved, the rook involved has never moved, the squares between the king and the rook involved are not occupied, the king is not in check, and the king does not cross over or end on a square in which it would be in check. If the move is valid, return true.

En passant is a special pawn capture which can occur immediately after a player moves a pawn two squares forward from its starting position, and an enemy pawn could have captured it had it moved only one square forward. If the move is valid, return true.

Case 2

Given a FEN string, generate the necessary data structures ("bitboards") to represent a chess board and all it's pieces. The complement is also tested: given a bitboard of the chess board, generate a FEN string.

Test [Pass]

Give a FEN string to a fenToPosition(FEN STRING) function, produce the correct bitboard. This will be shown by graphically printing the bitboard, and then comparing it against the FEN string.

Case 3

Given a piece that has been lifted from the board, calculate all the legal moves available. This is done by simulating the piece detector hardware.

Test [Pass]

Pass a bitboard representation of the board, and a specific piece to getMoves(BITBOARD, PIECE), and expect to get the right moves. This shall be tested for all pieces. Print a graphical representation of the available legal moves. Compare that to the expected result.

Had the hardware been connected, this information would have been passed to the LED controller.

Appendix D: README / Manual / Installation Guide for the Website

The README file below containing the Manual and the Installation Guide for the website is accesible online via: https://github.com/aninimus/boardcaster

Boardcaster (Website)

This is the website backend the powers the electronic chessboard with the same name developed for our Capstone course. Feel free to visit our live demo available at heroku. To access the hardware schematics and the firmware code you can just clone our bitbucket repository. You can also view more information in our about page, our blog and you can email us via our contact page.

Features

- User system (create, view, list, edit)
- Session management (login, logout)
- Game creation (create, view, list)
- Comment based feedback system (create, view, list)
- "Like" based feedback system (like, unlike)

Requirements

- Operating System capable of running Ruby on Rails, we recommend Ubuntu.
- Ruby >= 1.9
- Ruby on Rails >= 3.1
- See Gemfile for necessary ruby gems
- Git >=1.7 for Version Control

Installation

1. Clone the repository `git clone [url]`
2. `cd boardcaster`

3. `bundle install` to install necessary gems
4. `bundle exec thin start` or `rails server` to start the server
5. `open http://127.0.0.1:3000`

Author

Carlos Andreu <*carlos.andreu[@]upr.edu*> - Software Engineer from the University of Puerto Rico.

License

```
Boardcaster is a web application component of an electronic chessboard.
Copyright (C) 2011 Carlos Andreu

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see < http://www.gnu.org/licenses/ >.
```

Free Software, Yeah!

Appendix E: Budget Analysis Tables

Absolute Maximums

Total Projected Component Cost Base	$480.01
Parts Cost Overhead	60%
Absolute Cost Maximum	$768.02

Table 4: Absolute Maximums

Budget Analysis Summary

Final Parts Cost	$672.31
Projected Cost	$480.01
Absolute Cost Maximum	$768.02
Current Budget Status	-$192.3
Overhead Consumption	25.04%

Table 5: Budget Analysis Summar

Parts List

Flat Cable (1.27/unit) 8 units
Connector - 34p - Full Pitch (0.57/unit) 36 units
Reed Switches (0.68/unit) 70 units
Standard LED (0.20/unit) 70 units
Copper Plates
LEDs
Hall-Effect Sensors
Cables
Solder tin
Battery
Arduino Mega 2560
Wifi Shield
Router
1k Resistors
Multiplexers
Magnet Square
Decoders

Table 6: Parts List

Part Costs

Store	Cost
Mouser	$251.19
CuteDigi	$62.45
Ebay	$29.98
4PCB	$145.35
Radioshack	$12.95
Walmart	$28
Hardware Store	$4
Capri	$7.49
Sparkfun	$117.9
N/A	$13
Total	$672.31

Table 7: Part Costs

Note: These part costs include shipment totals for orders made from that store.

Employment Costs

Employment Cost	$10,616.51
Social Security Rate (6.20%)	$3,291.12
Healthcare (1.3% of salary)	$796.24
Transportation	$2,320.00
Catering	$3,480.00
Parts	$672.31
Labor and Parts Subtotal	$21,176.18
Estimated Overhead (179%)	$59,186.58
Grand Total	**$80,362.76**

Table 8: Employment Costs

Presentation

Our final presentation is accessible online [36] but it has not received a score as of December 17, 2011 at 4:00PM. Below is our presentation evaluation sheet:

Contents
Introduction/Background: __/5
Body __/5
Conclusion __/5
Subtotal __/15

Presentation Skills
Organization/Outline __/5
Appropriate to Audience __/5
Pronunciation, grammar, articulation __/5
Management of Questions __/5
Support of arguments with evidence __/5
Time Management __/5
Subtotal __/30

Overall
Overall Quality __/5
Knowledge of Material __/5
Subtotal Overall __/10

Subject Specific
Team is organized __/5
Presented Relevant Info. __/5
Timeline followed __/5
Completed work __/5
Assessment of Project __/5
Prototypes and Diagrams __/5
Subtotal Overall __/45

Total __/100