ENSC 351 Project Report Fall 2023

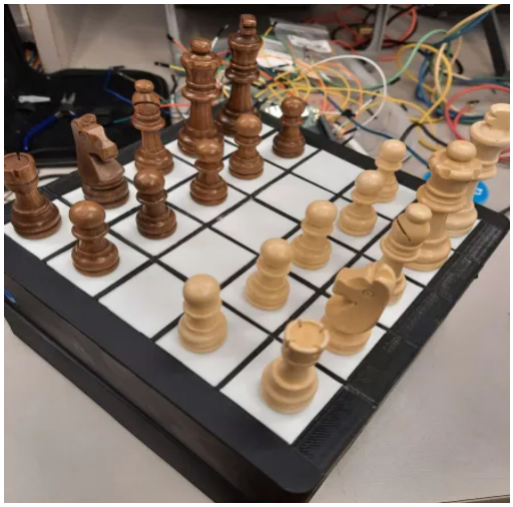
Group Name: SFUENSC10

Joel Lerner (jbl16@sfu.ca)
Lakshmi Narayanaprasath Kannan (lnk1@sfu.ca)
Xander Soriano (xbs@sfu.ca)
Ethan Price (erp3@sfu.ca)

System Explanation:

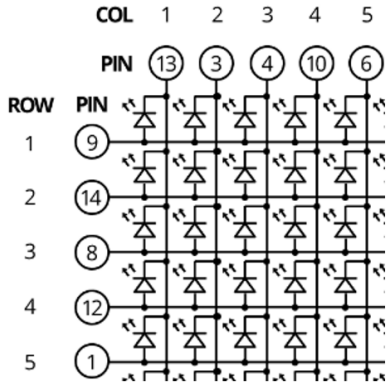
What does your system do? How does it do it? Include screenshots, pictures, and diagrams where possible.

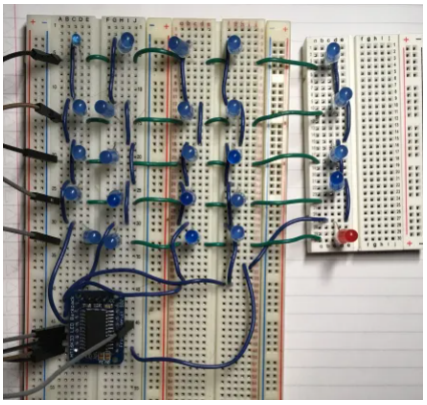
The system acts as a beginners 5x5 chessboard. While pieces are picked up, LEDS on certain tiles will light up to indicate the direction in which legal moves can be made. In order to do this, we will have LEDS attached to each tile on the chessboard as well as hall effect sensors which look for magnetic fields. We will lastly have magnets attached to the bottom of every piece, so the sensors can detect the presence or absence of each piece. The LEDS will be controlled by the 8x8 HT16k33 I2C backpack (multiplexer) used in Assignment 2, while the switches will be controlled and connected to a series of GPIO pins on BeagleBone Green.



LED Matrix

We wanted an LED on every tile on the 5x5 chessboard, meaning we created/wired our very own LED matrix on a breadboard while hooking up the LEDs to each tile. Below is a wiring diagram of the LED matrix and a prototype we made on the breadboard.

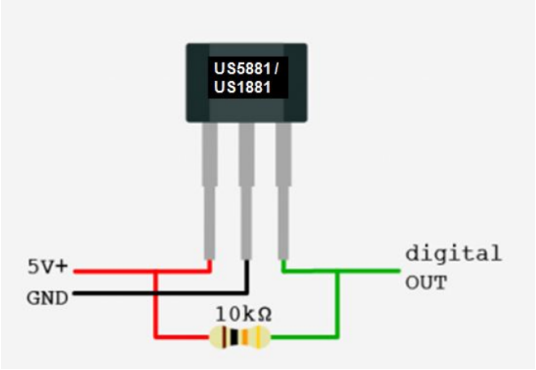




All of these LEDs are controlled by the 8x8 HT16k33 I2C backpack (multiplexer) used in Assignment 2. This module allows us to pass a 64 bit binary string to the matrix. Every 8-bits represent a single row on the matrix where a 1 would indicate on and 0 would indicate off. The use of this backpack (multiplexer) greatly reduced the wire as without it, each LED would require 1 GPIO pin.

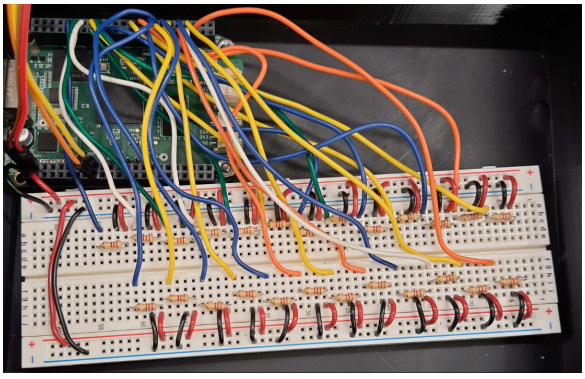
Hall Effect Sensor Matrix

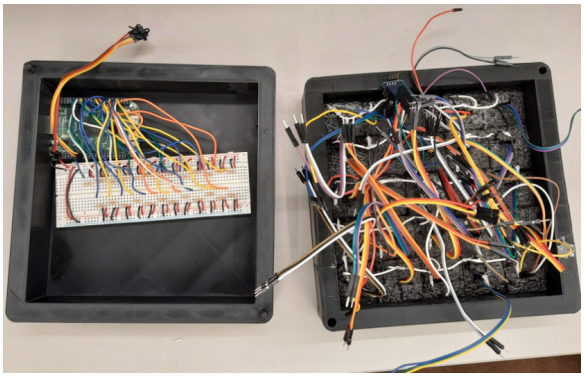
In order to detect the presence of pieces we used hall effect sensors which look for magnetic fields. If a magnetic field is detected the switch closes and current can flow through otherwise it will not. This means, when a magnet is placed close to the sensor the GPIO pin associated with the sensor will read 1 otherwise it will read 0. In order to wire the hall effect sensor we used the following schematic along with a pull up resistor in order to reduce electrical noise.



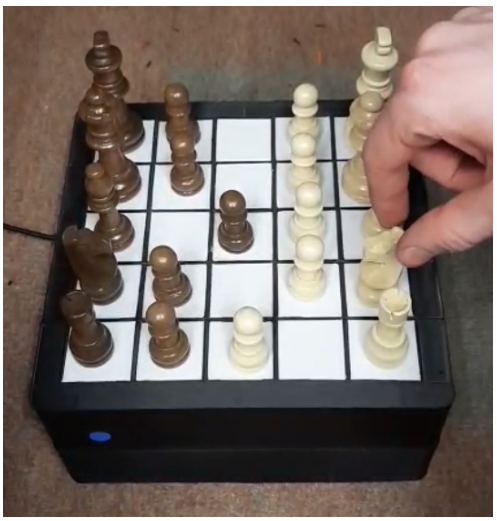
Similar to the LED matrix, we wanted a hall effect sensor on each tile. Therefore, we wired/added the sensors to the LED matrix. The sensors would be glued to the chessboard tiles while its leads would be configured to individual GPIOs on the BeagleBone and ground. The new matrix wiring is exactly the same with additional resistors connected along with every sensor.

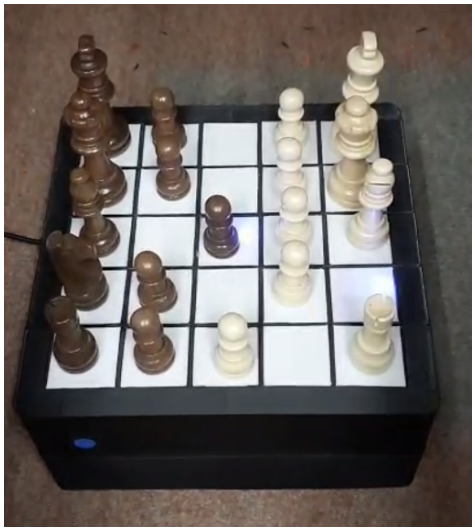
Below is a wiring image of the pull up resistor network used with the hall effect sensors and an image of the interior of the chess casing. The wiring is similar to that of the prototype image above but has longer wires to allow the hall effect sensors and LEDs to protrude through the floor of the chess board.





Below is the image of a player picking up the white horse. As shown in the image to the right, the corresponding tile the horse can move to are lit up





Chess Engine

Programmed in C, we wrote a chess engine that would emulate an NxN chessboard. Although we chose N=5 due to the complexity of wiring, the engine works up until N=8. The program tracks turns, tracks player moves, checks for possible moves, prevents invalid moves, allows capturing pieces and looks for checks and checkmates. In order to do this, we created a 2D array of tile structs. Inside each tile struct is a string which indicates the piece, and an integer that indicates the value of the hall effect sensor attached to the tile. We used the convention of lower case strings indicating black pieces and uppercase to indicate white. Afterwards, we implemented possible move logic for each piece. Most of these functions include a simple calculation that finds the indices of the tiles a piece can move to, a quick check if that move is actually valid or not , and a state change. For example, if there is an ally piece on a tile you can move to, that tile should not light up. Another example is a move that puts your king in check is not allowed. Furthermore, when you actually make a move the state should transition from a picked up to a waiting state. The last step was to implement capturing logic, which was an extension of the movement logic with an additional check. In order to capture a piece, you must pick up your own piece, then pick up the piece you want captured, then put your current piece on the new empty tile. This order is necessary since our program is searching for tiles that read sensor values of 0 (meaning picked up) and have an enemy piece on them. After this condition is met, capturing is as simple as replacing the target tile string with the new piece string.

Our main function acts as a high level wrapper for initializing the GPIOS and chessboard, reading and updating the sensors, running the chess engine, and exiting the program. It also creates and terminates two threads used for continuously reading/updating the hall effect sensors and updating the chessboard state.

Note: This is an extremely brief overview, more information is available in the source code.

State important things not working well (such as video streaming at .5 FPS, or bad sound, or flaky web page) and briefly explain what the team did to try and resolve these issues. A project can still be great while having some remaining challenges to solve.

Reed Switches

Initially reed switches were used instead of hall effect sensors to detect the presence of a magnetic field. We thought that since it had 2 leads instead of 3 it would be easier to use and wire. Similar to the hall effect sensor, these close when a magnetic field is detected. However, our initial tests showed that the reed switches were extremely fragile, short range and unresponsive. They would break extremely easily and required contact with the magnet to close. They would also sometimes stay shut even when the magnet was pulled away or stay open when the magnet was placed in contact. In order to resolve this issue, we switched over to hall effect sensors which required a pull up resistor. This made the sensors extremely responsive as well as improved range.

Feature table: (1-2 pages) Table of important product features

Description	Host/Target	Comp	Code	Authors	Notes
Hall Effect Sensors	T	5	C	Ethan/Joel	The hall effect sensors on each tile successfully detect when a piece is picked up. Sends a 0 if picked up else 1 to the BeagleBones GPIOs.
LED Matrix	T	5	C	Prasath/Joel	The LED Matrix successfully lights up the possible moves a piece can make when picked up.
Chess Engine	T	5	C	Ethan/Xander/ Prasath	The software successfully tracks turns, tracks player moves, checks for possible moves, prevents invalid moves, allows capturing pieces and looks for checks and checkmates.
Magnetic Chess Pieces	T	5	N/A	Joel	Pieces have magnets strong enough to trigger sensors under the chessboard.
Reset Button	T	5	C	Joel/Ethan	Reset button will reset the board state and turns off all LEDs.
Hall Effect Sensor Threading	T	5	C	Ethan/Xander/ Prasath	Hall effect sensors are successfully read in a separate thread than main.

Chess Engine Threading	T	5	C	Ethan/Xander/ Prasath	The chessboard is successfully updated and tracked in a separate thread than main.
Chess Board (physical)	T	5	N/A	Joel	Physical chessboard hides the wiring and lights up when pieces are picked up.

Extra Hardware & Software Used: (~1/2 page)

Beyond the BeagleBone, Zen Cape, and the course's guides/notes, what hardware or software libraries did you use (if any)?

Hall Effect Sensors

As mentioned numerous times above, we used hall effect sensors to detect the presence of magnetic fields. These were configured on the LED matrix breadboard with additional pull up resistors while being connected to the BeagleBones GPIOs.

3D Printer for ChessBoard and Case

We were able to use Fusion to create a 3D model of a simple 5x5 chessboard and a box that would hold all the wiring and BeagleBone. Afterwards, we printed the the models using a Polylactic Acid (PLA) filament.