Team AC: Smart Chess Board

Chris Lee, Austin Milford-Rosales, David Hanna

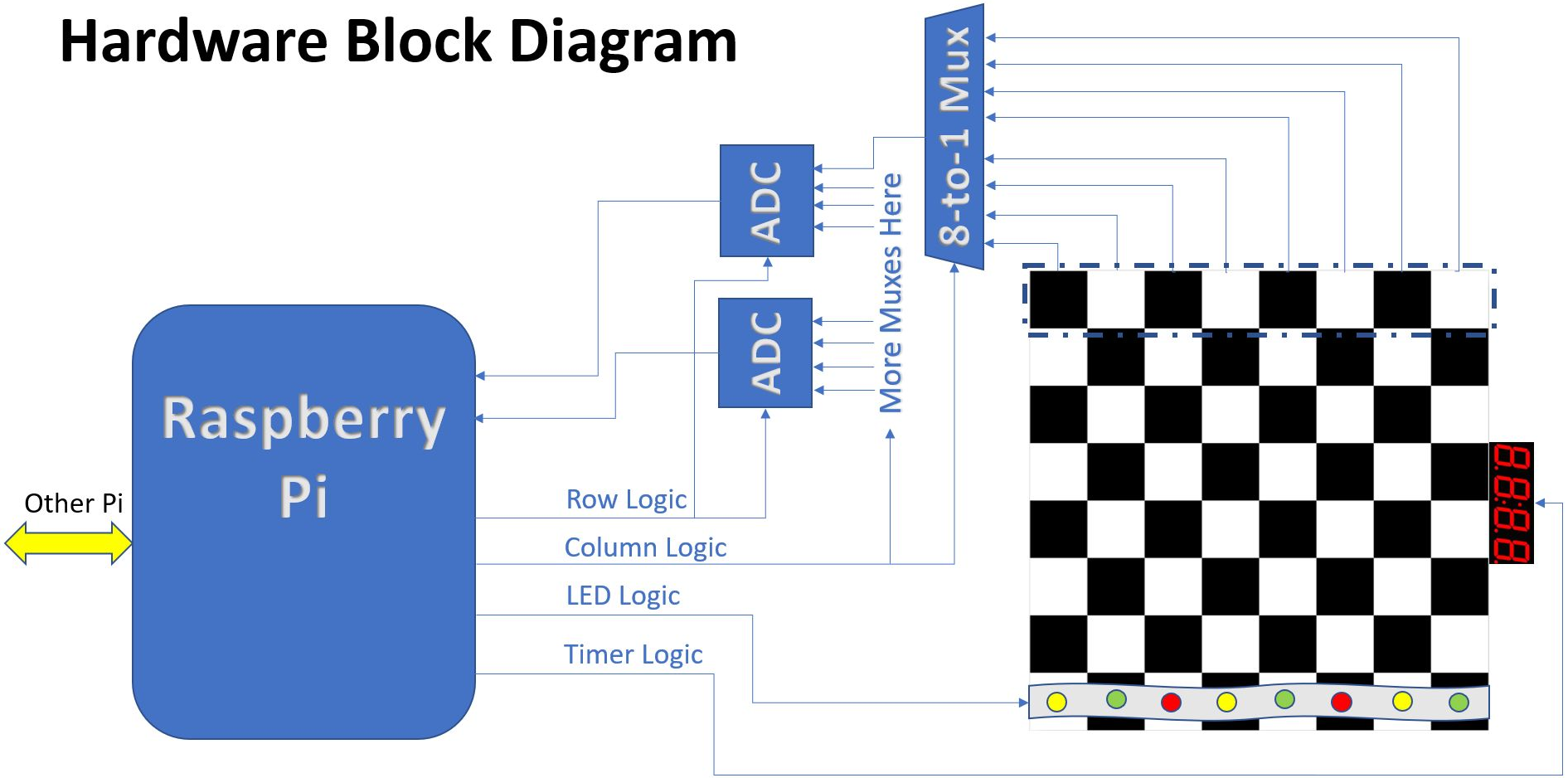
Application Area

- We wanted to make an accelerator. But... not within collective skillset :(
  - David found this: https://gitlab.com/fabriciorit/OpenGPU, gallium3d softpipe with hardware rasterizer plus block diagrams
  - Linux drivers + Graphics + RTL
- We decided to pivot to a Smart Chess Board!
- Motivation: We like playing chess on a real board, but don't always have physically present friends to play against
- This chess board will be able to do the following:
  - Identify which pieces are on which squares
  - Light up squares appropriately to signal intended moves, legality of moves, etc.
  - Automatically operate a chess timer
  - Connect to another board to enable live play between remote players

Solution Approach

- Each square will have an RGB LED to signal any information about that square to the player and a magnetic sensor (leaning towards the TI DRV5053OA) to determine what piece is placed on the square
- The board surface will be made of acrylic, and will have a box underneath to hide the greater circuit containing the sensor array and pi the clock itself will be sticking out the side
- A raspberry pi will be used to process data from the sensor array and run the game logic, and to interface with/run the remote game
- Our MVP will be connecting this board to another pi, which will run the game remotely. Ideally, we will have a second board connected to this pi as well.

Hardware Block Diagram

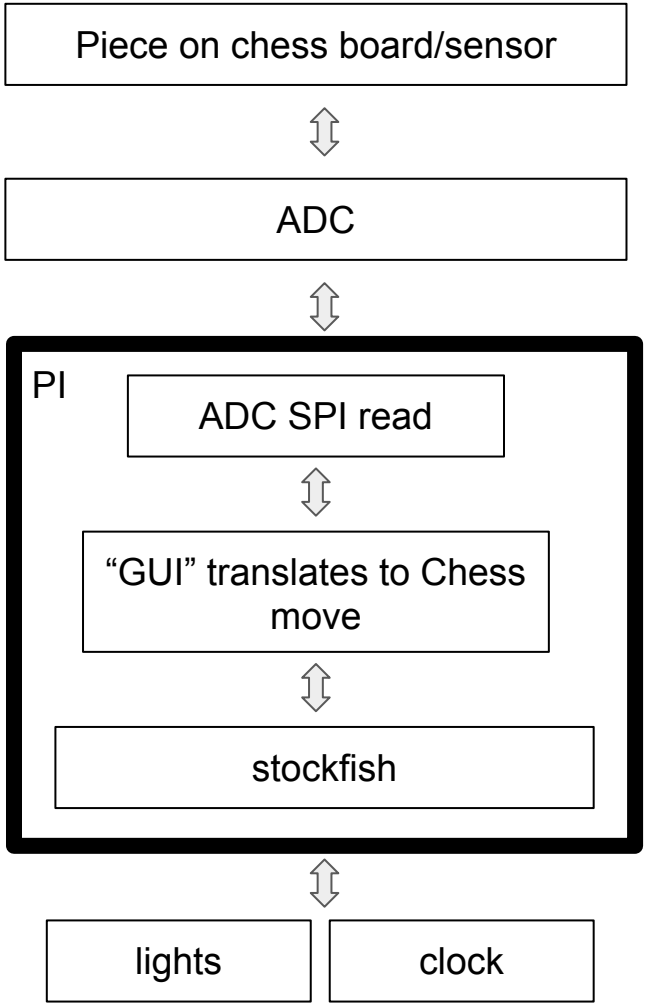


Hardware Overview

- Hall Effect Sensors
  - TI DRV5053OAQLPG
  - Can sense fields on the order of 10 mT
- Magnets
  - 1”x1”x1/16” N42 magnets
  - Can generate fields on the order of 10 mT with 1/8” separation
- 8-to-1 Mux
  - TI CD74HC4051-EP
  - Is a digitally controlled analog mux, takes 8 inputs and sends one output
- RGB LED strip
  - WS2812 RGB LED Strip
  - Can be controlled with 2 pins on the pi, lights are spaced out well and individually segmented

Software Overview

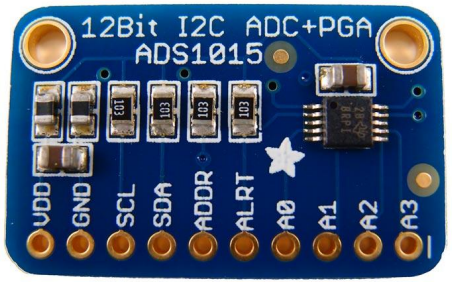
1. Player lifts piece
2. Pi samples Hall sensors via ADC
3. Pi interprets change
4. “GUI” translates change in board to (custom) UCI command
5. Stockfish returns legal moves based on game state
6. Custom logic lights RGB array



Software - ADC, Smart Clock

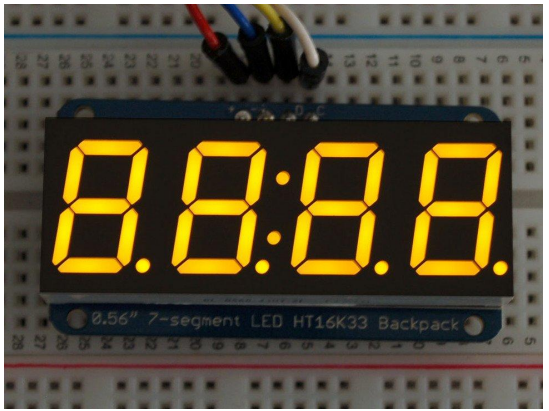
Sensor to ADC

- Hall effect sensors only provide analog output
- ADC required: ads1015 (4 inputs)
- Pi sweeps through array of 64 sensors
- Compares previous state of game with current game to determine piece lifted



Smart Clock

- Adafruit 0.56" 4-Digit 7-Segment Display w/I2C Backpack
  - Minimizes number of pins required
  - Already using i2c to communicate with adc
- Started when player lifts piece, stops when piece placed back on board

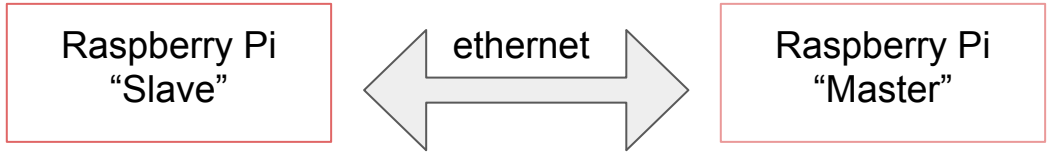


Stockfish Integration

- Stockfish as “chess engine”
  - Leading open-source chess engine
  - API based off of UCI
  - Clean C++ codebase
- Implements move generation and AI
- Expects a “GUI” that generates UCI commands
- Our “GUI” equivalent will be our smart chessboard + software that translates sensor data to chess moves
- Will need an actual custom debug GUI for our board
- Note: UCI = Universal Chess Interface, communication protocol for chess engines to communicate with user interfaces

Networked Play

- Two physical boards
- Pi's connected over ethernet
- Stockfish routines used on both Pi's for move/position validation
- "Master/Slave" configuration akin to Peer-2-peer video games



- Opens socket to master
- Read and interprets sensor data
- Send's legal moves to Master

- Owns game state
- Runs local server
- Reserves right to reject slave's input
- Read's and interprets boards sensor data

Testing Methods

- Sensor and square circuit testing
  - We will wire up one sensor through the ADC to the pi, then attempt to identify the following
  - A single sensor can react appropriately to the presence/lack of presence of a magnet
  - A single sensor can identify six different magnet strengths
- Board functionality testing
  - We will have a testing GUI implementation that will allow us to see the pi's interpretation of the current game state
  - Will help us debug errors in both piece placement and lighting of the full board
- Remote multiplayer testing
  - Fuzz slave input to ensure gracefully handling of errors
  - Hash out gameplay invariants to implement in host server

Metrics and Validation

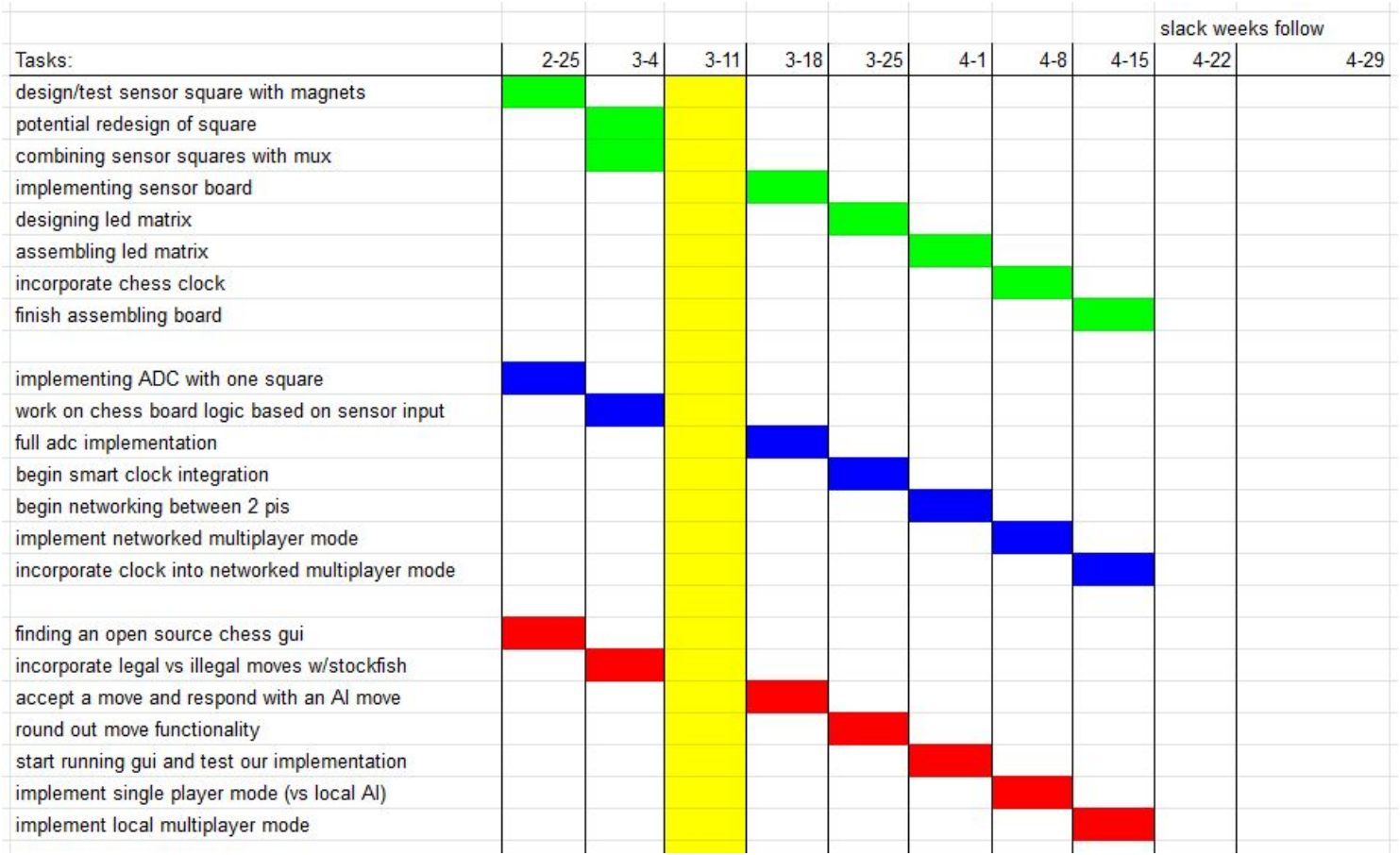
Potential areas for latency

- Determining which piece was picked up
  - Sweeping through the sensor array
  - Checking the piece type
- Running stockfish on the pi

Things to consider:

- Decision making process in chess
  - Player picks up piece, considers possible outcomes, puts piece down
  - Need to determine which piece was moved and possible next moves during this time
- Average human takes 0.25 seconds to recognize visual stimulus
- Board should react to a piece’s removal in a reasonable time, human visual reaction time of 0.25 seconds may be ideal

Gantt Chart


slack weeks follow
Tasks:
2-25
3-4
3-11
3-18
3-25
4-1
4-8
4-15
4-22
4-29
design/test sensor square with magnets
potential redesign of square
combining sensor squares with mux
implementing sensor board
designing led matrix
assembling led matrix
incorporate chess clock
finish assembling board
implementing ADC with one square
work on chess board logic based on sensor input
full adc implementation
begin smart clock integration
begin networking between 2 pis
implement networked multiplayer mode
incorporate clock into networked multiplayer mode
finding an open source chess gui
incorporate legal vs illegal moves w/stockfish
accept a move and respond with an AI move
round out move functionality
start running gui and test our implementation
implement single player mode (vs local AI)
implement local multiplayer mode