6.111 Final Project Proposal - Draft

ChessAi

Abstract

The goal of this project is to create a workable chess opponent who can be played against with little to no human computer interaction. Physically it consists of a chess board with 64 LED indicator lights, a set of standard chess pieces modified to be conductive on bottom, and some amount of wiring underneath the board to read off piece positions. In hardware on the FPGA will be an FSM to check the validity of all moves and to relay moves made by the human to a computer running a python library to determine the opponent's move. It will also interface with the LEDs to provide useful signaling and indications of the opponent's move.

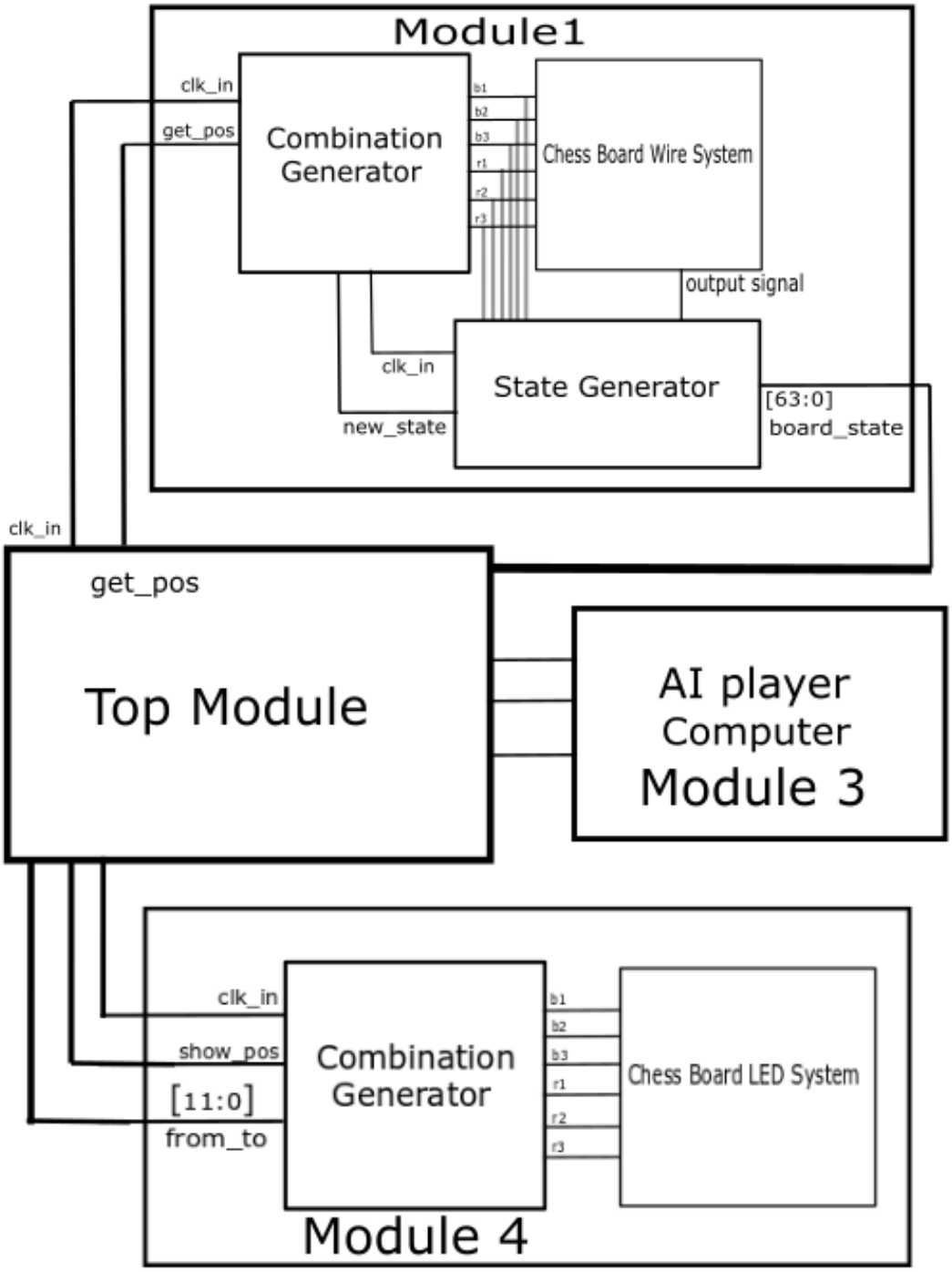
Project parts:

- Figure tracking via wires and a mux
- FSM for the state of figures
- Communication with the computer and the computer script
- LED matrix signaling opponent's movements

INTRODUCTION

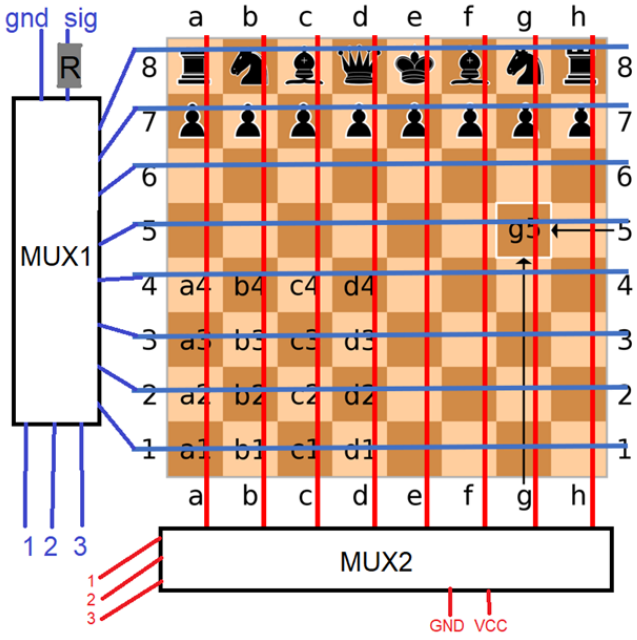
Many people love to play chess but nowadays it's hard to find people to play with – especially for the senior and more experienced population. A subgroup of those people either don't own or don't know how to use a computer, so they cannot play chess online. We decided to build a physical chess board that brings the joy of moving the figures with your hands and the pleasure and flexibility of playing against a computer.

THE BIG BLOCK DIAGRAM



MODULE 1 - The position of figures

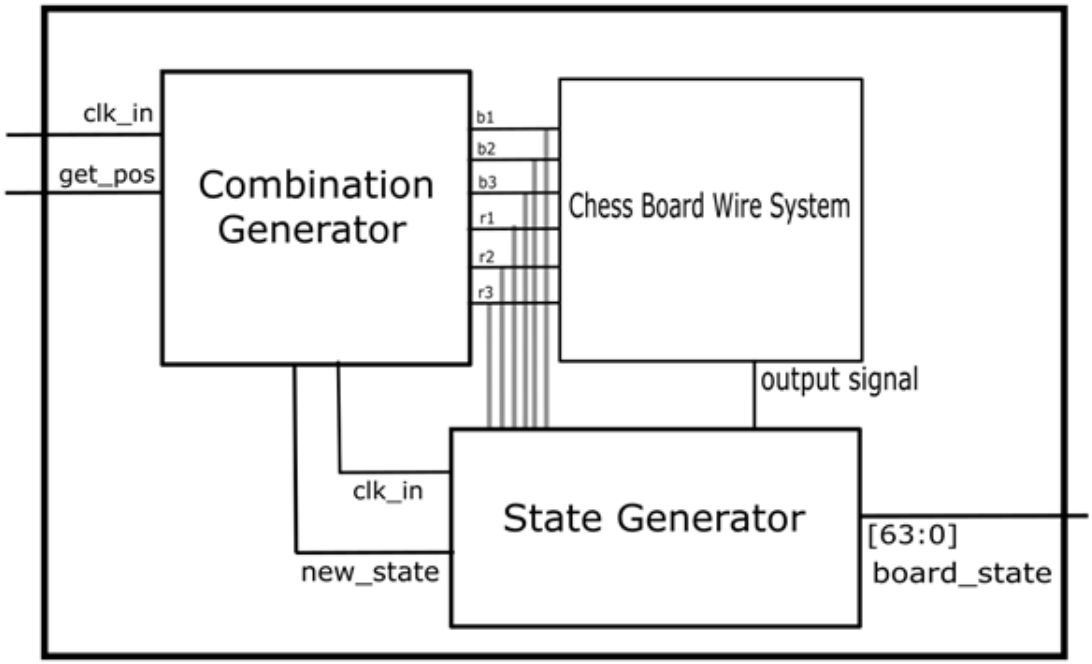
As we always need to know the position of every figure, we will build a module to do so. We will have 8 horizontal (blue) and 8 vertical (red) wires running under the board with exposed connections to the surface of the board. Whenever a figure (with a bottom plated with a conductor) is placed on a square, it will short the corresponding vertical and horizontal line. The lines will be connected to 2 multiplexers that will select which lines to analyze. The setup is shown in the figure below. MUX2 will individually supply the lines a-h with a high signal. At the same time, MUX1 will run through lines 1-8 and output a high signal if there is a figure on the corresponding square and a low signal otherwise. Since each mux is connected to 8 wires, we need only 3 bits to control each of them.



A submodule will wait for a get_pos signal and then run through 64 permutations activating and deactivating vertical and horizontal wires. The corresponding high and low signal will be used by another module that will generate a 64-bit long number that shows which squares of the chess board have figures on them.

The latency of this module is 64xSQUARE_WAIT (number of clock cycles spent looking at every square) We need to wait several clock cycles on each connection in order to debounce signal. This module can have a large latency (<10ms) since timing is not crucial in the whole system.

This module can only detect the presence/absence of figures on each square. In order to track different figures, we need an FSM to keep a record of the state of the game.



MODULE 2 - Chess FSM (also, top module)
Includes: Keeps track of figures, detects changes, tests legality of moves, communicates with other modules and looks out for the winner

While the essentials of chess basically only requires an FSM which keeps track of when a piece disappears and where it reappears, there are many many subtleties in both the rules and how the moves are enacted which will need to be considered since we want as little of the computation to be done on the off board computer as possible.

First the FSM needs to ensure that the move order is preserved. This means that if a black piece is picked up after another black piece and before a timer runs out (in the case of capturing or castling) a warning signal is flashed on the LED's.

Next, once the piece has been placed, the FSM must ensure that the move was a valid one. Naively, this should be an easy task, but again, chess is a complex game with complex rules. Pawns can only move forward once...unless they've reached the other side...or they can move twice if they're on the second row...well except if an opponent could capture them in the intermediate position (en peasant). There are a lot of these oddities which will need to be accounted for in the FSM.

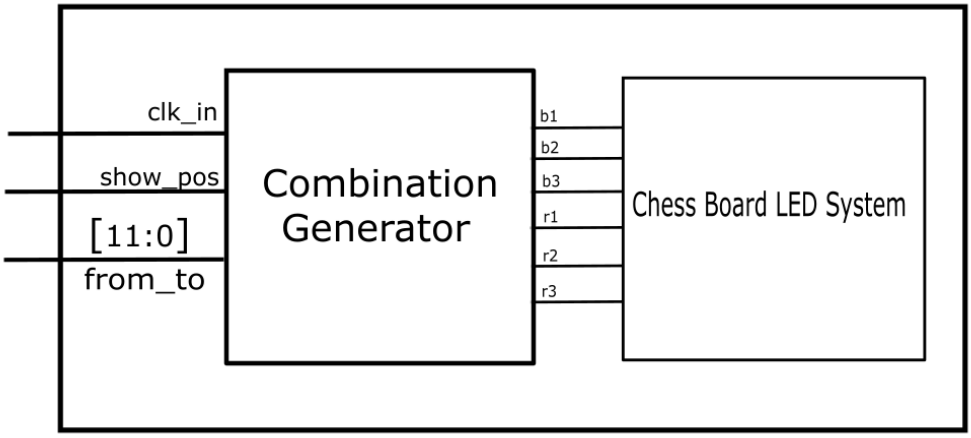
Once the movement has been verified by the FPGA, it will update the state of all the pieces accordingly then send the most recent move to the computer (over UART?) and wait for a response.

MODULE 3 - Response of the opponent
Includes: communication with the computer + python script
Ex: send:c2c4: returns: d2d4
Due to the complexity of chess gameplay and how much research and effort has been put into making decent chess AI on standard PC architectures, we have decided to leave the opponent move generation to a separate computer running some sort of standard chess library in python. This python script will simply receive moves generated by our system's FSM and respond to them with optimal opponent moves over UART. Communication will be artificially limited to single moves only to preserve the integrity of the project as a feat of hardware and not software.

As a secondary implementation, we could add some small amount of data exchange to configure the software at the beginning of the game. (Ex. sw[2:0] controls AI difficulty)

MODULE 4 - Opponent's movement signaling via LED's

Since this game is meant to be played against a computer - but on hardware, the simplest way for the computer to control the chess figures is to signal where the figures should be moved (by the player) via means of lighting up LEDs embedded in the chess board. Similar to the Module 1, we will have an additional set of 8 horizontal and 8 vertical wires running under the board. They will be able to individually power each of the 64 LEDs. Two LEDs will be turned on and off periodically: the one from where the figure is to be moved and the second one to where it should be moved. Again, a pair of 8-bit mux-es will be used to turn on/off individual LEDs.



One input to this module is the show_pos signal to start displaying the needed movement of the figures. The from_to signal is a 12-bit long signal. First 6 bits are designated to signal the coordinate of the “from” LED while the last 6 bits are to signal the “to” LED. The two LED’s will turn on and off alternatively until the player moves the figure.
Special function: if the from_to signal is 12’b1111_1111_0000 the LEDs will create an X sign (invalid move)!
Special function: if the from_to signal is 12’b1111_1111_1111 the LEDs will create nice patterns since the game is over!

***Maybe* MODULE 5 - On screen displaying of the game**
Includes: VGA output, gets data from the FSM

***Maybe* MODULE 6 - Camera tracking**
Includes: Makes module 1 obsolete :(

***Probably Not* MODULE 7 - Make opponent moves physical**
Includes: Physical actuators a la Etch-a-sketch with some electromagnetic tech.