6.111 Project Proposal

Rod Bayliss III and Brandon John

Gim Hom

29 October 2019

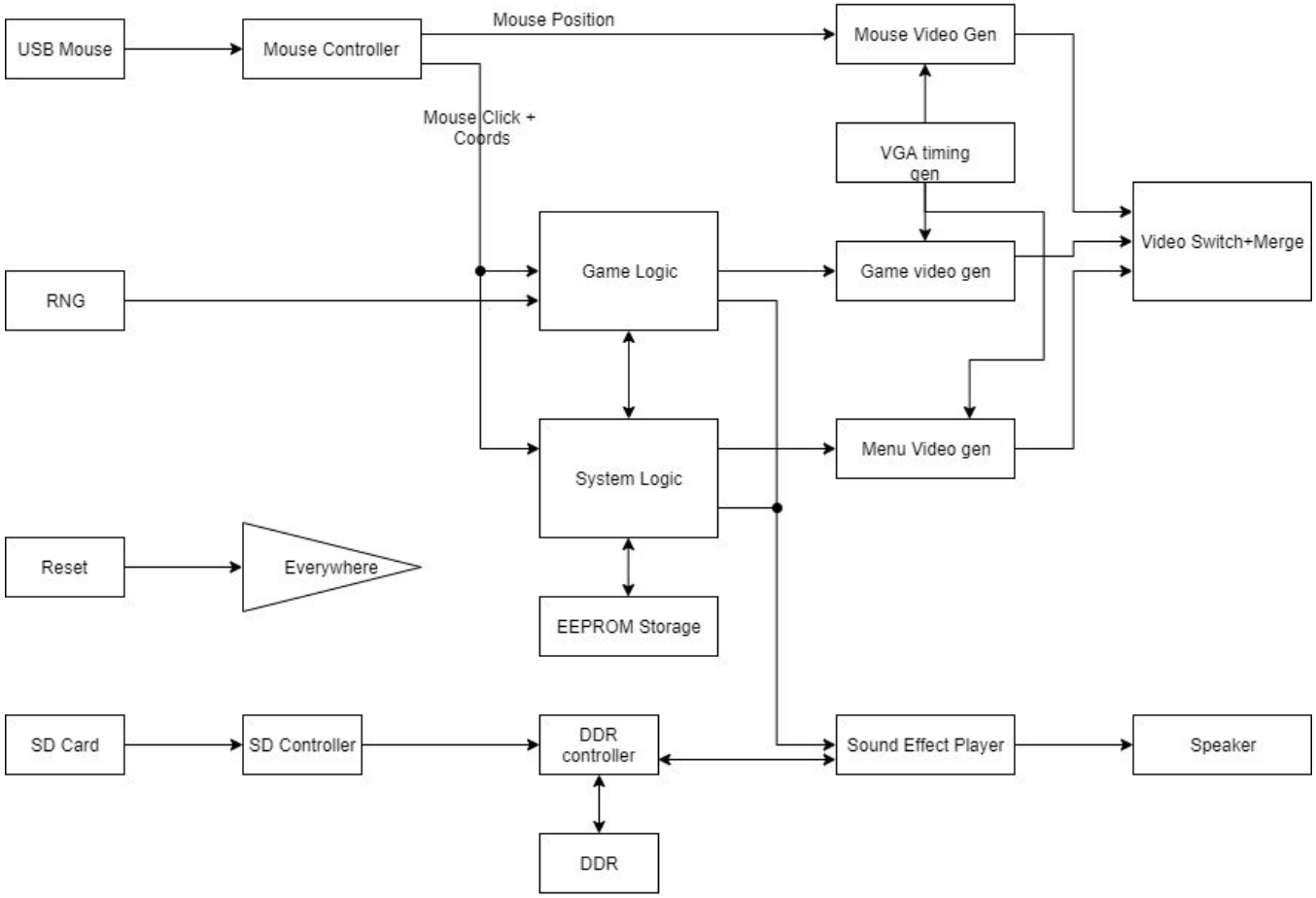
Abstract

Our project will be to create the classic arcade game “minesweeper” on an FPGA. Minesweeper traditionally involves a grid upon which the player can click to guess whether a cell is an empty cell or contains a mine. If they click on a mine, then the game is over. If they click on an empty cell, the cell will reveal how many neighboring cells have mines.

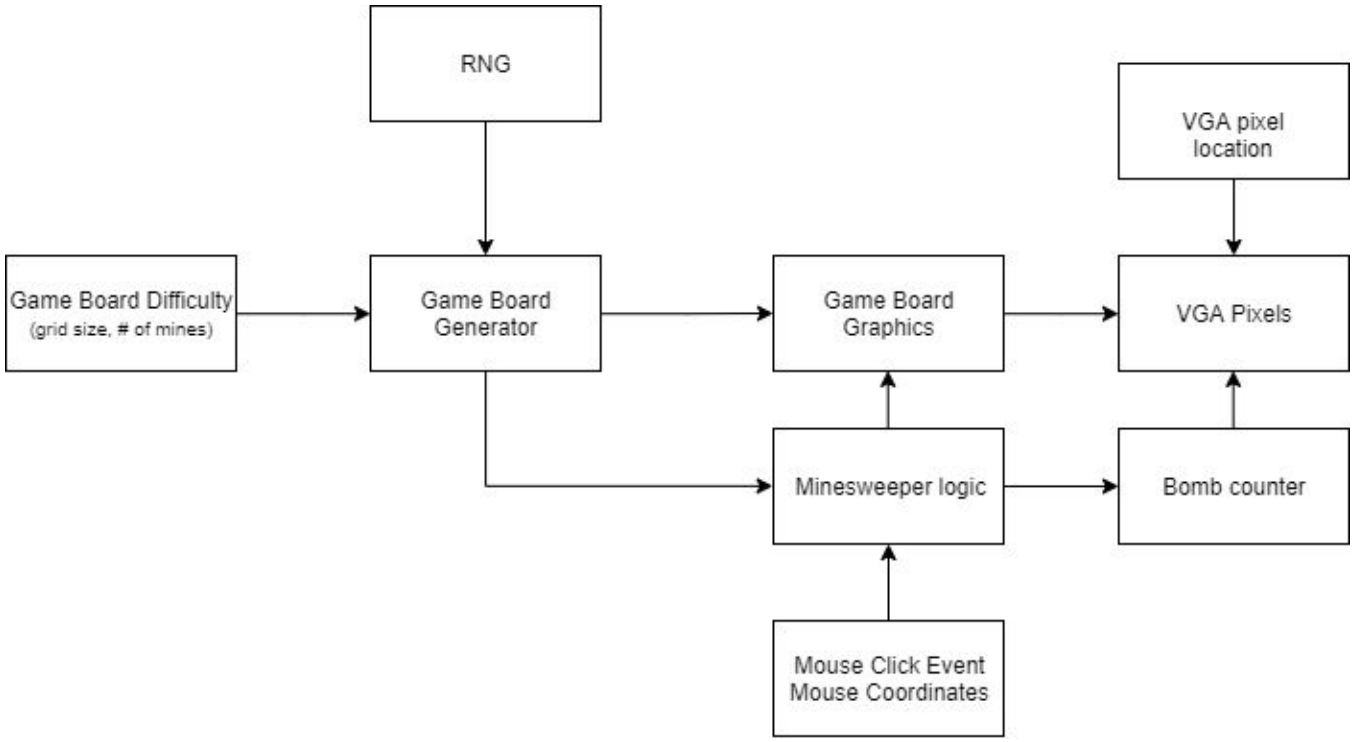
This FPGA implementation will involve several core modules: VGA or HDMI to draw and interact with the game, FSMs to run the game logic, sound effects through piezoelectric speakers, and a USB-HID mouse to interact with the game. As stretch goals for the project we hope to use SD card or flash storage to create a leaderboard of fastest times by level difficulty and potentially ethernet to have several independent FPGAs communicate their leaderboards and combine them or have multiplayer minesweeper (race to who can clear the board fastest, or have one person place bombs and the other clear, etc.)

Block Diagram

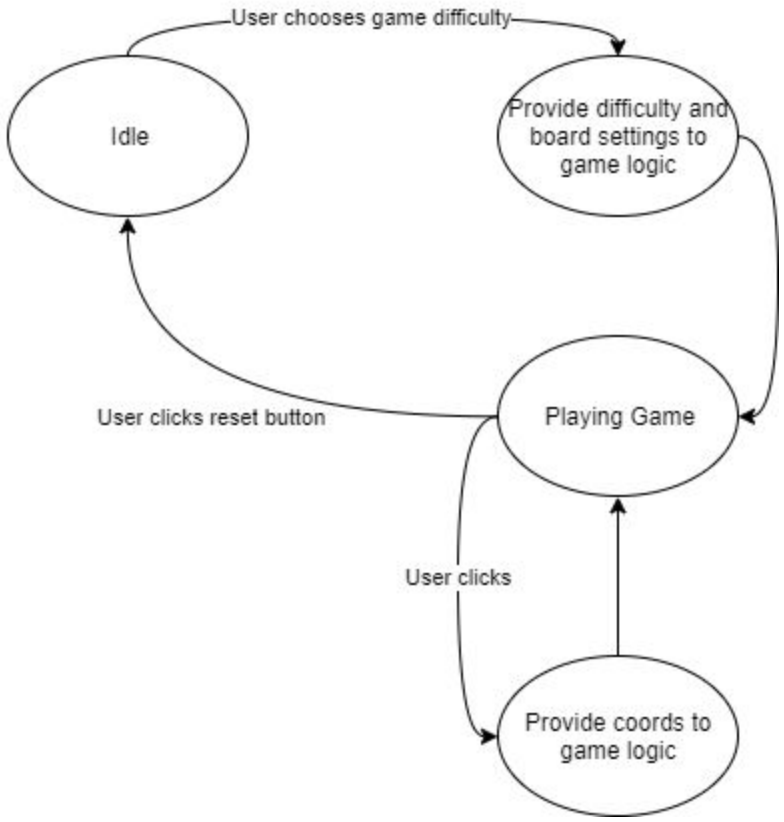
Inputs



Game Logic Diagram



System Logic FSM



System IO:

The system will only need a modest amount of IO to support this project. On the input side we will have a USB mouse (plugged into the USB Host port), an SD card (into the SD card slot), and a source of randomness (likely achieved by sampling the LSB of an ADC connected to “something”, perhaps the microphone that is built into the Nexys board? exact implementation TBD with some experimentation). The outputs are similarly straightforward, we will output audio over via the 3.5mm headphone jack and video through the VGA port (or potentially the HDMI port of the PYNQ board, depending on which we use). Beyond the traditional IO, we also will need to interface with a few on board memories, specifically the DDR memory and EEPROM storage.

Other hardware required:

We will only require a standard USB mouse, VGA display, and a speaker with a 3.5mm plug.

Modules + Interfaces:

The above block diagrams detail how this system will fit together. We have included a detailed list of every high-level module below, though please be aware that some of these modules may be further divided based on their complexity when we start implementing them. For each module, we have labeled who is in charge and included a brief description of the module

Mouse Controller (Brandon)

- Inputs: USB data
- Outputs: Current mouse position, events for mouse clicks
- Description: Implements USB-HID, listens for events from external mouse, accumulates movements, potentially includes acceleration and non-linear movement. Will have a multiplier or two, but this can easily be pipelined and still likely be faster than the screen refresh rate.
- Test Plan: While some test benches will be possible, a lot of this one will require just plugging in a mouse and watching the received decoded packets, probably via the 7 segment displays.

Mouse Video Gen (Brandon)

- Inputs: Current mouse position, current screen pixel
- Outputs: pixel color, boolean “override” control
- Description: Outputs the shape of a mouse onto the screen, with a bit set when the current pixel is covered by the pointer icon such that the final display mux will draw the pointer on top.
- Test Plan: Can be fairly easily tested with a testbench, and also integrated with just the mouse controller and a simple vga generator

Random Number Generator (Brandon)

- Inputs: 1 LSB from some analog input (slow to update)
- Outputs: 16 bit random number
- Description: Generates a random (16 bit?) number every clock cycle. Maybe every 2nd or 4th, depending on implementation details. Initial seed data TBD, but likely from the LSB of some sort of analog input.
- Test Plan: Record some outputs, make sure they are not the same between power cycles, and do not follow an obvious pattern.

SD Controller (Brandon/open source)

- Inputs: <Command to read a file>
- Outputs: data + address + source file indicator
- Description: Many exact details TBD based on what open source libraries we are able to integrate. We don't want to write our own SDIO stack, or even our own FAT controller, but are planning on writing the code to integrate them and to then read data from a file on the SD card and send it to the DDR controller.
- Test Plan: TBD based on what open source code we find. High level is just make sure that we can read a file by writing some checksum based on the contents of the file to the LCD.

DDR Controller (Brandon/open source)

- Inputs: write data + write address + write strobe + read address
- Outputs: read data + read strobe
- Description: We hope to find a library that handles the low level DDR interface, allowing us to focus on the interesting glue logic that writes data to the onboard DDR memory and reads it back as needed.
- Test Plan: write data to DDR, read it back and check that its correct. More details TBD based on what we have to write ourselves and what is available already.

Sound Effect Player (Brandon)

- Inputs: Begin Sound Effect trigger (1 clock pulse + ID of which effect to trigger)
- Outputs: 48kHz 8 bit audio to send to DAC or PWM module
- Description: Has 1 "player" per sound effect that can run simultaneously. Reads sound effect data from ram, saves in local cache as needed, then plays back. Will have resource conflicts to consider (multiple playback engines but only 1 ram, etc.), and will certainly need pipelining.
- Test Plan: Playback portion can be covered pretty well with test benches, but it will definitely need some isolated testing with hardware since it interfaces with two separate physical devices.

VGA Timing Generator (Rod/open source)

- Inputs: 65 MHz clock, pixels from modules
- Outputs: Draws the screen
- Description: XVGA module from lab 3. Generates vga control signals and receives pixel inputs from various modules to drive the screen

Game board generator (Rod)

- Inputs: Game board difficulty, RNG, 1st user click
- Outputs: Bomb location, minesweeper grid
- Description: Before the game starts, the user chooses the game difficulty which dictates the size of the grid and the number of bombs to be cleared. On the start click of the game, the minesweeper grid is generated, placing the bomb in a randomized location. One constraint is that there must be a clearing of tiles around where the user clicked (i.e. the first user click can't be a bomb and the game is more enjoyable if the first click generates a clearing)
- Test Plan: given an RNG input and number of bombs, spread out the bombs somewhat randomly. Difficulty/Grid size selector should be trivial

Game logic <main gameplay code> (Rod)

- Inputs: Mouse click event and pointer coordinates
- Outputs: Updated grid (to be fed into game board graphics module)
- Description: This modules task is determine what to do upon a user click on the game board. If the user clicks a bomb tile the game ends, if the user flags a tile the game board is visually updated and the flag counter is decremented. Finally, if the user clicks a clear tile this module determines how many surrounding tiles can also be cleared.
- Test Plan: Start from a very simplified game board and feed in mouse events and ensure the game board array is updated correctly.

Game video generator (Rod)

- Inputs: Game board array from Game logic
- Outputs: pixels
- Description: Given the game board array and game difficulty, this module draws the tiles on the screen and updates on command from the game logic module

System video generator (Rod)

- Inputs: Game status (flags, time), game difficulty, mouse event
- Outputs: Sidebar with the above information, user reset to system logic
- Description: this module draws the status bar in minesweeper. This allows the user to restart the game, view time left, number of flags left to place, and set the difficulty of the game.

Timer (Rod)

- Input: System Clock
- Output: Low frequency game clock
- Description: Outputs the clock for sidebar and feeds the final game time to record in the leaderboard.

Scoreboard (Rod)

- Input: Score at end of game
- Output: Current leaderboard
- Description: Keeps track of the game's leaderboard. This includes both the game difficulty and the time to clear. Will write to the SD card or EEPROM to maintain the leaderboard even after power-off.

Timeline

11/1: Proposal conference complete, preparation for design presentation begins
11/5: Project design presentation
11/8: Module interfaces written in verilog, high-level test benches created.
11/15: Game logic working for single difficulty. USB-HID module working
11/22: Game logic + video gen, USB-HID, VGA fully working at single difficulty
11/29: Multiple difficulties, SD-Card working, audio finishing up, leaderboard and end-game sequence finishing up
12/6: Project completed. Additional features potentially implemented such as multiplayer, snowflakes on screen, etc.
12/11: Final report