The Game of Life
DESIGN DOCUMENT

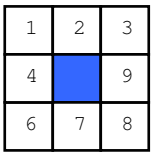
Steven Chen
Juan Gutierrez
Vincenzo Zarrillo
{stc2104, jmg2048, vaz2001}@columbia.edu

INTRODUCTION

The goal of our project is to design a visualization of the Game of Life as described by John Horton Conway.[1] The Game of Life was meant to show what happens to organisms when they are placed in close proximity to each other. Upon giving the Game initial conditions, each successive 'generation' (iteration) shows the evolution of the organisms. The Game is represented on a grid of squares with colored squares denoting a living organism.

GAME OF LIFE ALGORITHM

Each 'generation' (iteration) of the game updates the current status of the location of living organisms. Each organism has eight neighbors as illustrated below.

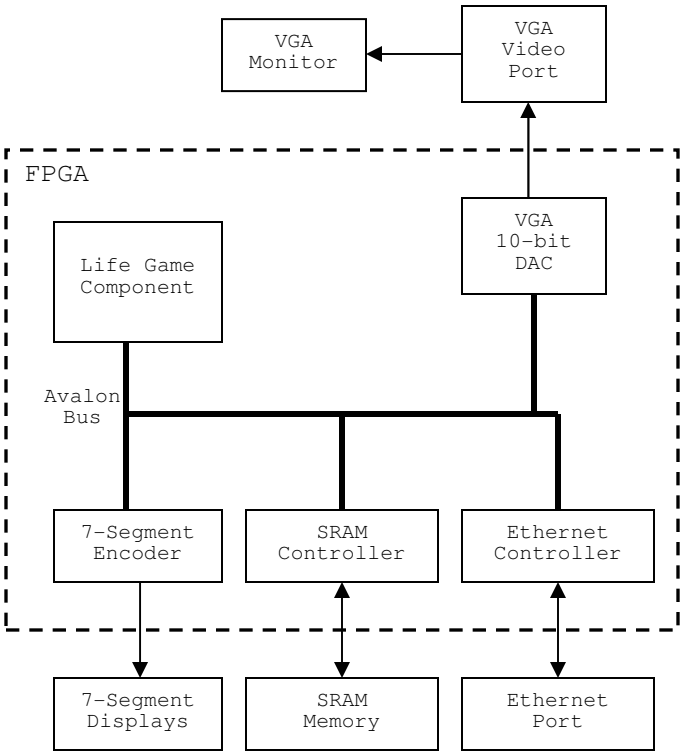


A living organism continues to live if it has either two or three neighbors which are also living. A empty area brings to life a new organism if it has exactly three neighbors. The following pseudocode illustrates the algorithm.

```
foreach (square in the system)    // run for entire grid
{
    count = 0;

    foreach (neighbor square)     // for each of the 8 neighbors
    {
        if (currentSquareState == alive)
            count++;
    }

    if (square is living)
    {
        if !( (count == 2)||(count == 3) )
            kill square;
    }
    else    // square is dead
    {
        if (count == 3)
            revive square;
    }

}
```

[1] Wikipedia article on the Game of Life: http://en.wikipedia.org/wiki/Conway%27s_game_of_life.

BLOCK DIAGRAM



DESIGN COMPONENTS

Life Component

- Receive initialization coordinates from either the internet via the Ethernet controller or from a C program (both using the Avalon Bus to send and receive data).
- Begin simulating game algorithm on start set of coordinates given.
- There will be two versions of the board saved in the FPGA memory: current_state and next_state. The current_state board is evaluated (with all resulting decisions about whether or not a being survives, dies, or is created in a square going into the next_state board).
- The end of a generation occurs when the last square in current_state is examined. Here, next_state is set to current_state, and next_state is reset. current_state is always the board being displayed by the VGA peripheral.
- The bits in the (current/next) state logic vectors will represent one square on the board. The [row,column] coordinates of current_state will be used to calculate the starting position of the square that the bit in current_state represents.
- Each time a generation cycle has been completed, a signal will also be sent to the LED component, triggering it to increase the display of the generation counter.

7-Segment Encoder Component

- Receives from the "life component" an "increase" signal, which will cause it to increment the number displayed on the hexes, representing which generation the game is currently on.

If we can: Genetics Component

- Intermediary component between "life component" and VGA peripheral.
- Each square will come with a basic set of "genes" (which will be depicted by the type of color they are – we haven't decided if we want the user to be able to define these things, or if we'll just assign them upon creation of generation zero).
- As dead cells produce new living organisms, the parents involved will donate some of their genes (the top half of each of its VGA_R, VGA_G, and VGA_B bits), and the other will donate their genes (the bottom half), thus producing a cross.
- Ideally we'd like to see certain combinations showing simulations of "dominant genes" taking over certain sections of generations.