Design:
Cellular Automata on FPGA

James Olsen (jco2127)
Serena Shah-Simpson (ss4354)
Jeff Tao (jat2164)
Robert Viramontes (rsv2111)
Priscilla Wang (pyw2102)

1

Introduction

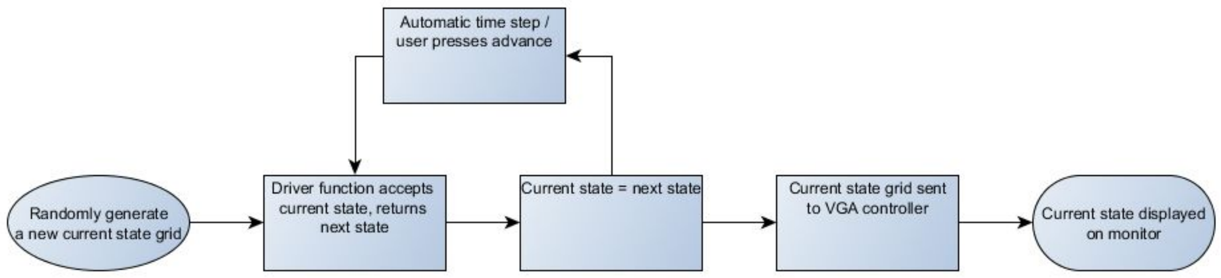
For our project, we will implement Conway's Game of Life cellular automata on our development board and display the results to a monitor. A user will be able to start the simulation to run automatically, step manually through generations, and restart with a new initial state. Next states will be computed by a hardware block on the FPGA, returned to the software, and then displayed via a VGA controller on the FPGA.

1 Image depicting Conway's Game of Life: https://i.ytimg.com/vi/QT_pKNzOOhQ/maxresdefault.jpg

Software Architecture

Main controlling game

The game will be controlled by software in userspace. This software will have a grid for the current state and a grid for the next state. The program will use a random number generator (if this is not available through C libraries on Linaro Linux, then we can use a random number generator IP core on the FPGA; this will require adding driver support) to fill the initial grid with either dead or alive cells. When the next state needs to be generated, we will invoke a function in the accelerator driver that accepts the current state grid and the size of the grid and returns the next state of the grid. We will then send the next grid to the VGA controller to draw to screen.


Automatic time step / user presses advance
Randomly generate a new current state grid
Driver function accepts current state, returns next state
Current state = next state
Current state grid sent to VGA controller
Current state displayed on monitor

GUI

In order to increase the visibility of the cellular automata, we will treat each 2x2 square of pixels on-screen as a single cell (a single pixel would be difficult to resolve for the observer or gain meaningful data from). This means that the largest grid we can display is 640x512. However, we would prefer to make this square (512x512), which will give us space to display commands. These commands will be entered via keyboard and captured by software (can use Lab 2 basics to interpret keyboard input) and will control the game to either start playing automatically (at a predetermined time step), manually (press “next” arrow key), or restart (generate a new random grid).

Accelerator driver

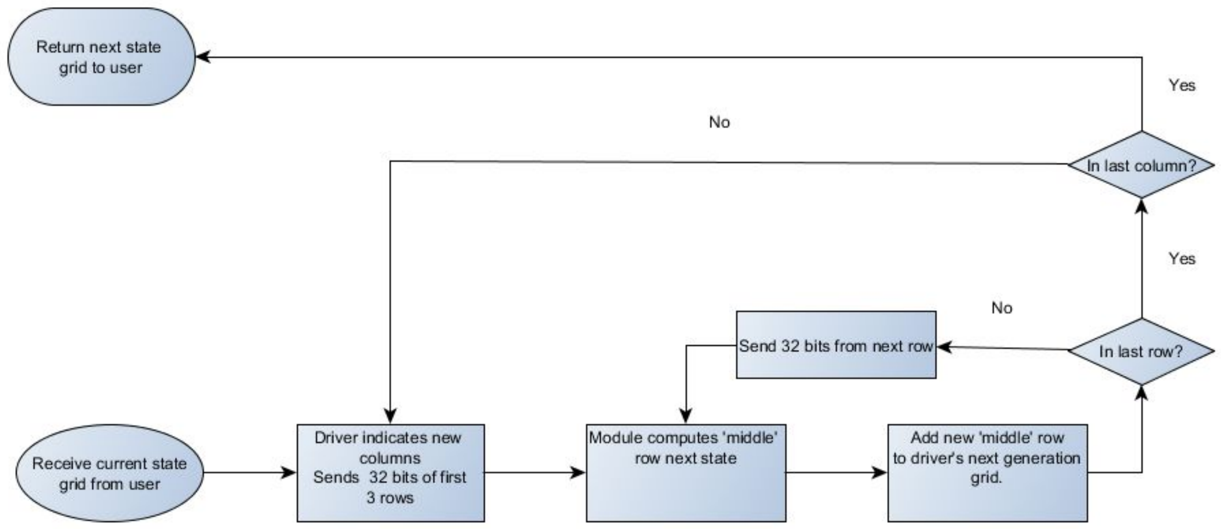
The next state of any cell depends on the surrounding 8 cells and its own current state, as follows[2]

1. Death: if the count is less than 2 or greater than 3, the current cell is switched off.
2. Survival: if (a) the count is exactly 2, or (b) the count is exactly 3 and the current cell is on, the current cell is left unchanged.
3. Birth: if the current cell is off and the count is exactly 3, the current cell is switched on.

Thus, to compute the next state of a given row, we need three rows worth of information. In order to take advantage of reusing data, we will compute entire rows of width 32 (the maximum

[2] http://mathworld.wolfram.com/GameofLife.html

iowrite size). For edges, we will treat them as if the grid is surrounded by ‘dead’ cells (i.e. for the first row, we will actually send it as the first “middle” row with a row of zeroes on top). The driver will receive the entire 512x512 grid from the user (in the main program described above) and manage the dispatching of the correct data to the accelerator module. It will collect the responses from the module and add them to an internal grid for the next state. When the grid is complete, it will return it to the user program.


Return next state grid to user
Yes
No
In last column?
Yes
No
Send 32 bits from next row
In last row?
Receive current state grid from user
Driver indicates new columns Sends 32 bits of first 3 rows
Module computes 'middle' row next state
Add new 'middle' row to driver's next generation grid.

<u>VGA driver</u>

The VGA driver will receive the entire current state grid and send this information to the VGA controller. It will be responsible for drawing the cells on-screen by mapping the 512x512 grid to the 1024x1024 pixels on screen and using the remaining columns on-screen to display the (static) user controls for the program.

Hardware Architecture

<u>Next State Accelerator</u>

We are able to accelerate the computation of the next state of a row because of the property that the next state of a single cell depends only on the current state of the cell and its surrounding neighbors, hence we can compute the next state for cells in parallel. While we could theoretically compute the entire grid in parallel, we avoid this for considerations of FPGA resources and particularly bus bandwidth (it would take many cycles to send this data back to the user, so the speedup of a fully parallel computation is buried). Below is pseudocode for determining the livelihood of a cell:

```
int i, j = 0;
sum = 0;
for ( ; i < 3; i++)
     for ( ; j < 3; j++)
          sum+=grid[x-1+i, y-1+j]
sum-=grid[x,y]
if (sum == 3) grid[x,y] = 1;
else if (sum !=2) grid[x,y] = 0;
```

Our design obtains 3 rows of length 32 each, so we will have the data to compute 30 new states each clock cycle. Once these are computed, we need to shift each row up one (middle becomes top, bottom becomes middle) as we receive the new bottom row from the driver. The driver sends the next row upon successful receipt of the newly computed state of the previous middle row. It can then proceed again on the new middle row. In this way, the module is only aware that it is computing a single row and the driver takes care of managing which rows are being computed (simpler to implement in C).

VGA Controller

For the VGA driver, we are planning on modifying the VGA driver developed in Lab 3 to support displaying at native resolution (1280 x 1024 pixels) with a refresh rate of 60Hz. This will require a pixel clock of 108 MHz[3], which we plan to generate using the system clock of 50 MHz and one of the PLL blocks provided on the board (configured through the Quartus Megawizard).

Memory Considerations

Our grid is 512 bits by 512 bits, or 32768B ~ 33KB. With a RAM of 1GB connected to the processor, we should have no problem storing the data required for the multiple copies and versions required in the user program and the driver.

Milestones

Milestone 1: Complete VGA driver running at native resolution and 60Hz refresh. Can test this with results from Lab 3. Implement a smaller, non-graphical version of Conway's Game of Life (i.e. print to console) to design and test the user controls.

Milestone 2: Update VGA controller and driver to display a grid and the user controls. Implement and test accelerator module with Conway's Game of Life algorithm.

[3] https://eewiki.net/pages/viewpage.action?pageId=15925278

Milestone 3: Complete drivers for VGA controller and accelerator. Update user program to be aware of and use the drivers for computation and display.

“Bonus” Feature:
Pattern[4] Counter: This cellular automata has a few, well-known patterns that develop. It could be useful to implement a feature that counts the number of these patterns currently on screen.

[4] https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life#Examples_of_patterns