The Game of Life

FINAL PROJECT REPORT

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

May 8, 2007
# Table of Contents

**Introduction** ................................................................................................................. 2  
**The Game of Life** ....................................................................................................... 2  
**Design and Implementation** .................................................................................. 3  
  Organisms and Game Board ...................................................................................... 3  
  Board Representation in System ................................................................................ 3  
  System Architecture Overview .................................................................................. 4  
  Game Logic ................................................................................................................... 6  
**Timing** ............................................................................................................................ 7  
**VGA** .............................................................................................................................. 8  
**NIOS Processor and Software** ................................................................................. 8  
**Role in the Group and Lessons Learned** ................................................................ 9  
  Steven Chen .................................................................................................................. 9  
  Juan Gutierrez............................................................................................................... 9  
  Vincenzo Zarrillo ....................................................................................................... 10  
**Code Listing** ................................................................................................................. 11  
  `vga_update.vhd` ........................................................................................................... 11  
  `hello_world.c` ............................................................................................................... 26
INTRODUCTION

The goal of our project is to design a visualization of the Game of Life as described by John Horton Conway. The Game of Life is 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 ‘board’ of the game is meant to represent the ecosystem in which the organisms live in. In Conway’s representation, this is in essence a large grid. Each box in the grid represents one organism. Colors are used to indicate whether alive or dead. The details of our board are outlined later.

THE GAME OF LIFE

The Game of Life is not really a game, but more of an algorithm. The only user interaction in the game is the passing of the user’s initial conditions to the game board.

Each organism on the board (represented by a square) has 8 neighbors as illustrated below:

```
1 2 3
4 5 9
6 7 8
```

In order to determine the state of the board on the next generation of the game, each organism is examined. A living organism continues to live if it has either two or three neighbors which are also living. A dead organism is brought to life if it has exactly three neighbors which are also living. In all other scenarios, the organism dies or continues to be dead.

Generations continue forever; however, this may not be noticed on the board as in many cases, the system reaches a steady state and no new changes to the board are introduced.

---

DESIGN AND IMPLEMENTATION

Organisms and Game Board

In order to keep as many components of our design in terms of powers of two, the size of our game board is 256x256 organisms. Since each organism is represented by one pixel, our game board is 256 pixels tall by 256 pixels wide.

In order to simplify our design when it comes to border conditions, we have designated the border around the entire board to be filled with dead organisms. Thus, our actual game board is 254x254, a 1.6% reduction from 256x256.

Board Representation in System

We represent each organism as one bit in RAM. The following diagram gives a representation of the board:

Each 'cell' holds 32 bits
8 cells X 32 bits = 256 bits total per row
8 cells X 256 rows = 2048 \( (2^{11}) \) cells total in board

Each of our cells in RAM holds 32 bits. Since our board is 256 bits in length and width, we can think of this as having 8 cells of 32 bits per row and 256 rows in total.
System Architecture Overview

Below is the system diagram of our project.

As you can see, the architecture of our project is very simple. The Nios Processor is responsible for sending the user specified initial conditions to the vga_raster through the Avalon bus. The following is a more detailed look at the vga_raster.
The vga_raster in essence serves three functions. The first is to accept the initial conditions from the Nios Processor across the Avalon bus. The second is to take the current state of the board and output it to the screen through the ‘VGA’ block. The last, but equally important job of the vga_raster is to update the current state of the board and output the next generation to another block of RAM.

As you can see, our design contains two main RAM components, RAM 1 and RAM 2. These two components were generated using the MegaWizard function in Quartus. Below is a block diagram look at our RAM components.

In order to have a system where data could be read out of RAM to be updated and also be read out of RAM to be output to the screen, we used dual-port RAMs generated from the MegaWizard. The reason we had two RAMs was for double buffering purposes – one RAM was used to hold the next iteration while the other one was used to hold the current iteration. When the updater part of the vga_raster was done updating, a flag would be raised and the two RAMs would be swapped, making what was current the next RAM and vice versa.
Game Logic

The game logic in its entirety was done by the updater component of vga_raster. The following diagram illustrates an overview of how our system updates for the next generation:

There are 3 34-bit shift registers that are meant to hold the contents of one RAM 1 cell (as discussed earlier on). The 2 bits of excess are required for the next time the register is loaded with the adjacent cell’s bits. The reason we load three rows is because we need to examine the 8 surrounding neighbors of each cell in order to determine its life or death for the next generation. The variable countNeighbors holds the value of how many surrounding neighbors are alive (hold a value of 1). Depending on whether the current organism we are examining is dead or alive, the appropriate next generation value is written to a 32-bit register. Upon filling the 32-bit register with data, this register is then written to RAM 2.
TIMING

Initially, the VGA is outputting nothing to monitor (nothing in terms of the board – the whole monitor outputs blue waiting for the first RAM to be loaded and read). Software passes 32 bits to each address space on the Avalon bus. In software, because Avalon runs on a 50 MHz clock and VGA runs on a 25 MHz clock (our RAM’s are implemented in the VGA raster and connected to the same clock), there is a delay between software writes, to make sure enough clock cycles have passed by so that the data is securely passed from the 50 MHz Avalon bus to the 25 MHz RAM.

After software passes the last address to hardware, the raster then begins reading from the first RAM. There is a displayRate signal defined in hardware, which counts up to about 3 million clock cycles, and then calls for an update to occur. While the update is occurring, VGA continues to read from the same RAM (due to our dual port RAM implementation). The reading that VGA does requires that just before we finish looking at the last bit, we need to load the next address from RAM so that we output the right bit that corresponds to the current pixel position.

When an update needs to occur, it is activated when the VGA has reached the last address of the first RAM (when we’re no longer on the board). We needed to read in three rows at a time and because there are 8 addresses of 32 bits in each of the 256 rows, we first need to read 0, 8, and 16, write to 8 then, 1,9,17, write to 9, etc., etc. (the top row and bottom row were a border of zeros that we hard coded and the side columns were also zeros – hence we can just write to address 8 at the very beginning). With each read, we waited about 2 clock cycles AFTER we passed what we wanted to RAM (one cycle for RAM to get the address, one cycle to make sure RAM had enough time to send it out to its output bus).

The next part that occurs is the analysis of neighbors. At first we tried counting all the neighbors at once and that simply and frankly does not work. We check for one neighbor at a time and after searching the last neighbor, we look at the cell being examined and write the output bit to the outRegister (which will be written to the “next” RAM). This increased the updating time but it was not a problem because VGA checks if the updater is done, and if so, it swaps, but only when it wants to. So there is actually a segment of time where the updater is not doing anything because it has already completed its cycle, but VGA needs to finish reading and outputting. Therefore, just before you get the beginning of the board, VGA checks if updater has completed, and if so, the RAM swap occurs and VGA reads from the “next” RAM and updater will read from “next” RAM and write back to “current.”
VGA

The job of the VGA component of the vga_raster was to properly output bits from RAM onto the screen. Since our game board was 256px by 256px, we centered the board and told the raster that any pixels outside of the 256x256 board should be colored blue. Then in order to determine what organisms were alive or dead, we read bit by bit the contents of RAM. If the bit was 1, we would color the pixel white; otherwise, we could color it blue.

NIOS PROCESSOR AND SOFTWARE

This project did not make heavy use of software. Thus the Nios processor implementation along with the Avalon bus implementation was very much like that of lab 3. We set the initial conditions (patterns of bits) in our C program and then let the hardware take over. Of note is the following macro:

```c
#define IOWR_VGA_DATA(base,offset,data) IOWR_32DIRECT(base,offset,data)
```

This macro was used to write conditions directly to the hardware. For example,

```c
IOWR_VGA_DATA(VGA_BASE, 1044, 7);
```

would write the value 7 (represented as 0........0111 on the board) into address 1044.
ROLE IN THE GROUP AND LESSONS LEARNED

Steven Chen

Embedded System Design is really like no other class I’ve taken in my four years at Columbia. While the foundation of knowledge was laid in the computer science and electrical engineering classes I have taken, much of the learning in this class happens in the lab – writing VHDL code, looking at timing diagrams, drawing out designs, etc.

During this project, I was primarily responsible for designing and implementation of the Game of Life logic. This design underwent many ideas and changes and finally with the assistance of Professor Edwards and our TA Yingjian, we were able to come up with a reasonable design. Furthermore, I was also responsible for putting together the design document, final presentation and this report.

A key learning experience for me was to stop thinking about the way I programmed in software and think more about clock cycles and what can and should happen after every clock cycle. The timing analyzer proved to be one of the most important tools during our implementation. We cannot stress how important the analyzer was to our group. Also, with long compile times (well over 5 minutes) the process is a lot longer than in software so clean designs will save you much time compiling.

As with what most other students in this class will say, it is never too early to start working on this project. No matter how thought-out a design you have, you have probably left a crucial component out or have made over-simplifying assumptions that will render your design near useless. Thus, it is never too early to start working on this project.

Juan Gutierrez

I took on a lot of responsibility for this project. I worked on all coding aspects of the project, but focused mostly on the updater and the VGA. Along with Steve and Vinny, I spent most of the time debugging, looking at timing diagrams and redesigning parts of the project to ensure proper systems integration.

This class was a crash course in hardware. As a computer science major, I have never had to deal with clock cycles and timing. My advice to future groups is to start using the timing simulator early and often. The simulator will save you
the trouble of not knowing what’s going on and guessing what may or may not be happening. Without the simulator, there’s no chance we would have been half as successful as we were. As a group that ran into multiple design and implementation problems, my best advice would be to consult the professor and the teaching assistants for as much advice as possible because they probably know the best way to get a project done! Lastly, I’d encourage all future groups to test every possible condition before moving onwards. We ran into a condition that partially broke the system towards the end of the project and had a very difficult time looking for the source of the problem.

Vincenzo Zarrillo

With Juan, the main parts of the project I worked on was the Nios and Avalon aspects of the project in addition to the complete integration of the system – making sure the different parts of the system could talk to each other. However, most of the time I, along with the rest of the group, was designing and redesigning the different parts of the system and debugging any problems that would come up. This involved a lot of time looking at simulator diagrams, the most important tool of our project.

The main advice I’d give to future groups is the same advice I’ve seen past groups say in their reports: Start early! In addition, taking more time at the beginning to design the system and constantly talking to the TAs is a good way to go about this project. Also, like I mentioned earlier, the only way to make sure you have the timing right, which was crucial in a system like ours where a lot of bits were being moved around, examined, etc., is to use the timing simulator as much as possible.
-- Game of Live/VGA raster display
-- Juan, Steve, Vinny - Team 24

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity vga_raster is
  port (reset: in std_logic;
        clk25_diff : in std_logic; -- Should be 25.125 MHz
  HEX : out std_logic_vector(6 downto 0);
  VGA_CLK,        -- Clock
  VGA_HS,        -- H_SYNC
  VGA_VS,        -- VSYNC
  VGA_BLANK,     -- BLANK
  VGA_SYNC: out std_logic; -- SYNC
  VGA_R,         -- Red[9:0]
  VGA_G,         -- Green[9:0]
  VGA_B: out std_logic_vector(9 downto 0) -- Blue[9:0]
);

architecture rtl of vga_raster is
  component board_mem
    PORT
      (address_a : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
      address_b : IN STD_LOGIC_VECTOR (10 DOWNTO 0);

  generic
    (size : positive := 10);
    signal subreg : std_logic_vector(9 downto 0);
    signal subreg_in : std_logic_vector(9 downto 0);
    signal subreg_out : std_logic_vector(9 downto 0);
    signal subreg_address : std_logic_vector(10 downto 0);
    signal subreg_data : std_logic_vector(9 downto 0);
    signal subreg_chipselect : std_logic);

end component board_mem;

entity board_mem is
  PORT
    (address_a : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
    address_b : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
    size : positive := 10;
    subreg : std_logic_vector(9 downto 0);
    subreg_address : std_logic_vector(10 downto 0);
    subreg_data : std_logic_vector(9 downto 0);
    subreg_chipselect : std_logic);
end entity board_mem;

architecture rtl of board_mem is
  generic
    (size : positive := 10);
  signal subreg : std_logic_vector(9 downto 0);
  signal subreg_in : std_logic_vector(9 downto 0);
  signal subreg_out : std_logic_vector(9 downto 0);
  signal subreg_address : std_logic_vector(10 downto 0);
  signal subreg_data : std_logic_vector(9 downto 0);
  signal subreg_chipselect : std_logic;

begin
  subreg_in <= subreg;
  subreg_out <= subreg;
  subreg_address <= subreg_address;
  subreg_data <= subreg_data;
  subreg_chipselect <= subreg_chipselect;
end architecture rtl;
clock : IN STD_LOGIC;
data_a : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
data_b : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
wren_a : IN STD_LOGIC := '0';
wren_b : IN STD_LOGIC := '0';
q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
q_b : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
);
end component;

-- Video parameters
constant HTOTAL : integer := 800;
constant HSYNC : integer := 96;
constant HBACK_PORCH : integer := 48;
constant HACTIVE : integer := 640;
constant HFRONT_PORCH : integer := 16;

constant VTOTAL : integer := 525;
constant VSYNC : integer := 2;
constant VBACK_PORCH : integer := 33;
constant VACTIVE : integer := 480;
constant VFRONT_PORCH : integer := 10;

-- Hpixel_pos - starts at 0
signal Hpixel_pos : std_logic_vector(10 downto 0) := "00000000000";

-- Vpixel_pos - starts at 0
signal Vpixel_pos : std_logic_vector(10 downto 0) := "00000000000";

-- Signals for the video controller
signal Hcount : std_logic_vector(9 downto 0); -- Horizontal position
(signal Vcount : std_logic_vector(9 downto 0); -- Vertical position
(signal Hcount : std_logic_vector(9 downto 0); -- Horizontal position
(signal AddrToCurrent_a : std_logic_vector(10 downto 0) := "00000000000";
(signal AddrToCurrent_b : std_logic_vector(10 downto 0) := "00000000000";
(signal DataToCurrent_a : std_logic_vector(31 downto 0);
(signal JunkData : std_logic_vector(31 downto 0);
(signal AddrToCurrent_a : std_logic_vector(10 downto 0) := "00000000000";
(signal AddrToCurrent_b : std_logic_vector(10 downto 0) := "00000000000";
(signal AddrToNext_a : std_logic_vector(10 downto 0) := "00000000000";
(signal AddrToNext_b : std_logic_vector(10 downto 0) := "00000000000";
signal wrenToNext_a : std_logic := '0';
signal qFromNext_a  : std_logic_vector(31 downto 0);
signal qFromNext_b  : std_logic_vector(31 downto 0);

signal displayRate   : std_logic_vector(17 downto 0) :=
"000000000000000000";

-- game logic related
signal sr_counter      : std_logic_vector(3 downto 0)   := "0000";
signal sr_pos          : integer := 33;
SIGNAL countNeighbors  : STD_LOGIC_VECTOR(3 downto 0) := "0000";
--SIGNAL currentPositionInOutput : STD_LOGIC_VECTOR(4
downto 0) := "11111";
signal currentPositionInOutput : integer := 31;
signal beginningRow       : std_logic := '1';
signal rowStartCount      : std_logic_vector(1 downto 0) := "00";

signal pass : std_logic := '1';
signal pass_counter : std_logic_vector(3 downto 0) := "0000";

SIGNAL dataOut1,dataOut2,dataOut3 : STD_LOGIC_VECTOR(33 downto 0) := 
"0000000000000000000000000000000000";
SIGNAL outRegister       : STD_LOGIC_VECTOR(31
downto 0) := "0000000000000000000000000000000000";

signal address : std_logic_vector(10 downto 0) := "00000000000";

signal bitPos_ram : std_logic_vector(4 downto 0) := "11111";
signal hexToHEX   : std_logic_vector(6 downto 0) := 
"1111111";
signal bitPos      : std_logic_vector(4 downto 0) := 
"11111";
signal VGAReg      : std_logic_vector(31 downto 0);
signal videoBit    : std_logic := '0';
signal junkBit     : std_logic := '0';
signal load_new    : std_logic := '1';

--signal first_pass : std_logic := '1';
signal update      : std_logic := '0';
signal swap        : std_logic := '0';
signal done_update : std_logic := '0';
--signal load_next : std_logic := '0';

signal loadData_a, loadData_b, loadq_a, loadq_b : std_logic_vector(31
downto 0);
signal loadAddr_a, loadAddr_b : std_logic_vector(10 downto 0) :=
"000000000000";
signal loadwren_a, loadwren_b : std_logic := '0';

signal ram_address : std_logic_vector(10 downto 0) := 
"000000000000";
signal load_address : std_logic_vector(10 downto 0) := 
"000000000000";
signal ram_loaded : std_logic := '0';
signal loaded : std_logic := '0';
signal loadDelay : std_logic_vector(1 downto 0) := "00";
begin
loadRAM : board_mem PORT MAP
(  
clock => clk25_diff,
  data_a => loadData_a,
  data_b => loadData_b,
  address_a => loadAddr_a,
  address_b => loadAddr_b,
  wren_a => loadwren_a,
  wren_b => loadwren_b,
  q_a => loadq_a,
  q_b => loadq_b
);
currentRAM : board_mem PORT MAP
(  
clock => clk25_diff,
  data_a => dataToCurrent_a,
  data_b => junkData,
  address_a => AddrToCurrent_a,
  address_b => AddrToCurrent_b,
  wren_a => wrenToCurrent_a,
  wren_b => junkBit,
  q_a => qFromCurrent_a,
  q_b => qFromCurrent_b
);
nextRAM : board_mem PORT MAP
(  
clock => clk25_diff,
  data_a => dataToNext_a,
  data_b => junkData,
  address_a => AddrToNext_a,
  address_b => AddrToNext_b,
  wren_a => wrenToNext_a,
  wren_b => junkBit,
  q_a => qFromNext_a,
  q_b => qFromNext_b
);

load : process (avs_s1_clk)
begin
  if avs_s1_clk'event and avs_s1_clk = '1' then
    if avs_s1_reset_n = '0' then
      ram_loaded <= '0';
    elsif avs_s1_chipselect = '1' and ram_loaded = '0' then
      if avs_s1_write = '1' then
        loadwren_a <= '1';
        loadAddr_a <= avs_s1_address;
        loadData_a <= avs_s1_writedata;
        if avs_s1_address = "1111111111" then
          ram_loaded <= '1';
        end if;
      end if;
    end if;
  end if;
end process load;
-- Horizontal and vertical counters

HCounter : process (clk25_diff, reset)
begin
  if reset = '1' then
    Hcount <= (others => '0');
  elsif clk25_diff'event and clk25_diff = '1' then
    if EndOfLine = '1' then
      Hcount <= (others => '0');
      Hpixel_pos <= "00000000000";
    else
      if Hcount = HSYNC + HBACK_PORCH + 192 + Hpixel_pos then
        Hpixel_pos <= (Hpixel_pos + 1);
        end if;
        Hcount <= Hcount + 1;
      end if;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk25_diff, reset)
begin
  if reset = '1' then
    Vcount <= (others => '0');
  elsif clk25_diff'event and clk25_diff = '1' then
    if EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
        Vpixel_pos <= "00000000000";
      else
        if VCount = VSYNC + VBACK_PORCH + 112 + Vpixel_pos then
          Vpixel_pos <= (Vpixel_pos + 1);
          end if;
          Vcount <= Vcount + 1;
        end if;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk25_diff, reset)
begin
  if reset = '1' then
    vga_hsync <= '1';
  elsif clk25_diff'event and clk25_diff = '1' then
    if EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;
HBlankGen : process (clk25_diff, reset)
begin
if reset = '1' then
  vga_hblank <= '1';
elsif clk25_diff'event and clk25_diff = '1' then
  if Hcount = HSYNC + HBACK_PORCH then
    vga_hblank <= '0';
  elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
    vga_hblank <= '1';
  end if;
end if;
end process HBlankGen;

VSyncGen : process (clk25_diff, reset)
begin
if reset = '1' then
  vga_vsync <= '1';
elsif clk25_diff'event and clk25_diff = '1' then
  if EndOfLine = '1' then
    if EndOfField = '1' then
      vga_vsync <= '1';
    elsif Vcount = VSYNC - 1 then
      vga_vsync <= '0';
    end if;
  end if;
end if;
end process VSyncGen;

VBlankGen : process (clk25_diff, reset)
begin
if reset = '1' then
  vga_vblank <= '1';
elsif clk25_diff'event and clk25_diff = '1' then
  if EndOfLine = '1' then
    if Vcount = VSYNC + VBACK_PORCH - 1 then
      vga_vblank <= '0';
    elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
      vga_vblank <= '1';
    end if;
  end if;
end if;
end process VBlankGen;

updater : process (clk25_diff)
begin
if clk25_diff'event and clk25_diff = '1' then
  if ram_loaded = '0' then
    load_address <= (others => '0');
    loaded <= '0';
  elsif ram_loaded = '1' and loaded = '0' then
    if load_address = "11111111111" then
      loaded <= '1';
      wrenToCurrent_a <= '0';
      AddrToCurrent_a <= (others => '0');
    elsif loadDelay = "00" then
      wrenToCurrent_a <= '1';
    end if;
  end if;
end if;
end process updater;
AddrToCurrent_a <= load_address;
loadAddr_b <= load_address;
elsif loadDelay = "11" then
dataToCurrent_a <= loadq_b;
load_address <= (load_address + 1);
end if;
loadDelay <= (loadDelay + 1);
elsif update = '1' and done_update = '0' then
if swap = '0' then
  if(load_new = '1') then
    -- initial load
    if sr_counter = "0000" then
      address <= AddrToCurrent_a;
      if AddrToCurrent_a = "11111111000" then
        dataToNext_a <= outRegister;
        wrenToNext_a <= '1';
        AddrToNext_a <= "11111111111";
        AddrToCurrent_a <= "00000000000";
        done_update <= '1';
      end if;
      dataOut1 <= (others => '0');
      dataOut2 <= (others => '0');
      dataOut3 <= (others => '0');
      outRegister <= (others => '0');
      sr_counter <= "1111";
    elsif AddrToCurrent_a(3 downto 0) = "0000" or
        AddrToCurrent_a(3 downto 0) = "1000" then
      beginningRow <= '1';
      wrenToNext_a <= '1';
      dataToNext_a <= outRegister;
      outRegister <= (others => '0');
      AddrToNext_a <= (AddrToCurrent_a + "111");
    end if;
    elsif(sr_counter = "0011") then
      dataOut1(31 downto 0) <= qFromCurrent_a;
    elsif(sr_counter = "0100") then
      AddrToCurrent_a <= (address + "1000");
    elsif(sr_counter = "0111") then
      AddrToCurrent_a <= (address + "10000");
    elsif(sr_counter = "1011") then
      dataOut3(31 downto 0) <= qFromCurrent_a;
      AddrToCurrent_a <= address;
      load_new <= '0';
      wrenToNext_a <= '0';
      if beginningRow = '1' then
        sr_pos <= 31;
        currentPositionInOutput <= 30;
      else
        sr_pos <= 33;
        --currentPositionInOutput <= 31;
      end if;
    end if;
    sr_counter <= (sr_counter + 1);
  else
    sr_pos <= 33;
    --currentPositionInOutput <= 31;
  end if;
  sr_counter <= (sr_counter + 1);
else
if sr_pos = 32 and currentPositionInOutput = -1 then
    wrenToNext_a <= '1';
dataToNext_a <= outRegister;
AddrToNext_a <= (AddrToCurrent_a + "111");
    --outRegister <= (others => '0');
    pass <= '1';
currentPositionInOutput <= 31;
elsif sr_pos = 31 and currentPositionInOutput = 0 then
    --pass <= '1';
currentPositionInOutput <= 31;
elsif (sr_pos > 1) and pass_counter < 8 then
    if (dataOut1(sr_pos) = '1' and pass_counter = 0) then
        countNeighbors <= (countNeighbors + dataOut1(sr_pos));
    end if;
    if (dataOut1(sr_pos-1) = '1' and pass_counter = 1) then
        countNeighbors <= (countNeighbors + dataOut1(sr_pos-1));
    end if;
    if dataOut1(sr_pos-2) = '1' and pass_counter = 2 then
        countNeighbors <= (countNeighbors + dataOut1(sr_pos-2));
    end if;
    if dataOut2(sr_pos) = '1' and pass_counter = 3 then
        countNeighbors <= (countNeighbors + dataOut2(sr_pos));
    end if;
    if dataOut2(sr_pos-2) = '1' and pass_counter = 4 then
        countNeighbors <= (countNeighbors + dataOut2(sr_pos-2));
    end if;
    if dataOut3(sr_pos) = '1' and pass_counter = 5 then
        countNeighbors <= (countNeighbors + dataOut3(sr_pos));
    end if;
    if dataOut3(sr_pos-1) = '1' and pass_counter = 6 then
        countNeighbors <= (countNeighbors + dataOut3(sr_pos-1));
    end if;
    if dataOut3(sr_pos-2) = '1' and pass_counter = 7 then
        countNeighbors <= (countNeighbors + dataOut3(sr_pos-2));
    end if;
pas
    pass <= '0';
    pass_counter <= (pass_counter + 1);
elsif pass = '0' then
    if dataOut2(sr_pos-1) = '1' then
        countNeighbors <= (countNeighbors + 1);
    end if;
    if countNeighbors = "0010" or countNeighbors = "0011"
then
    --if beginningRow = '1' then
    --    outRegister(sr_pos-1) <= '1';
    --else
    --    outRegister(currentPositionInOutput) <= '1';
    --end if;
else
    --if beginningRow = '1' then
    --    outRegister(sr_pos-1) <= '0';
    --else
    --    outRegister(currentPositionInOutput) <= '0';
    --end if;
end if;
else
if countNeighbors = "0011" then
    --if beginningRow = '1' then
    outRegister(sr_pos-1) <= '1';
    --else
    outRegister(currentPositionInOutput) <= '1';
    --end if;
else
    --if beginningRow = '1' then
    outRegister(sr_pos-1) <= '0';
    --else
    outRegister(currentPositionInOutput) <= '0';
    --end if;
end if;
if sr_pos = 2 then
    wrenToNext_a <= '0';
end if;
countNeighbors <= "0000";
currentPositionInOutput <= (currentPositionInOutput - 1);
pass_counter <= "0000";
sr_pos <= (sr_pos - 1);
pass <= '1';
else
    if beginningRow = '1' then
        outRegister(31) <= '0';
    end if;
dataOut1(33 downto 32) <= dataOut1(1 downto 0);
dataOut2(33 downto 32) <= dataOut2(1 downto 0);
dataOut3(33 downto 32) <= dataOut3(1 downto 0);
AddrToCurrent_a <= AddrToCurrent_a + 1;
load_new <= '1';
pass <= '1';
address <= (others => '0');
pass_counter <= "0000";
beginningRow <= '0';
sr_counter <= "0000";
countNeighbors <= "0000";
end if;
elsif swap = '1' then
    --swap = '1'
    if(load_new = '1') then
        -- initial load
        if AddrToNext_a = "11111111000" then
            dataToCurrent_a <= outRegister;
            wrenToCurrent_a <= '1';
            AddrToCurrent_a <= "11111111111";
            AddrToNext_a <= "00000000000";
            done_update <= '1';
            dataOut1 <= (others => '0');
            dataOut2 <= (others => '0');
            dataOut3 <= (others => '0');
            outRegister <= (others => '0');
            sr_counter <= "1111";
        else
            outRegister <= (others => '0');
            wrenToCurrent_a <= '1';
            AddrToCurrent_a <= AddrToNext_a;
            AddrToNext_a <= "00000000000";
            done_update <= '1';
            dataOut1 <= (others => '0');
            dataOut2 <= (others => '0');
            dataOut3 <= (others => '0');
            outRegister <= (others => '0');
            sr_counter <= "1111";
        end if;
    end if;
end if;
end if;
elsif swap = '1' then
    --swap = '1'
    if(load_new = '1') then
        -- initial load
        if AddrToNext_a = "111111111000" then
            dataToCurrent_a <= outRegister;
            wrenToCurrent_a <= '1';
            AddrToCurrent_a <= "11111111111";
            AddrToNext_a <= "00000000000";
            done_update <= '1';
            dataOut1 <= (others => '0');
            dataOut2 <= (others => '0');
            dataOut3 <= (others => '0');
            outRegister <= (others => '0');
            sr_counter <= "1111";
        else
            outRegister <= (others => '0');
            wrenToCurrent_a <= '1';
            AddrToCurrent_a <= AddrToNext_a;
            AddrToNext_a <= "00000000000";
            done_update <= '1';
            dataOut1 <= (others => '0');
            dataOut2 <= (others => '0');
            dataOut3 <= (others => '0');
            outRegister <= (others => '0');
            sr_counter <= "1111";
elsif AddrToNext_a(3 downto 0) = "0000" or AddrToNext_a(3 downto 0) = "1000" then  
beginingRow <= '1';  
wrenToCurrent_a <= '1';  
dataToCurrent_a <= outRegister;  
AddrToCurrent_a <= (AddrToNext_a + "111");
end if;
elsif(AddrToNext_a(3 downto 0) = "000 0" or AddrToNext_a(3 downto 0) = "1000") then  
beginingRow <= '1';  
wrenToCurrent_a <= '1';  
dataToCurrent_a <= outRegister;  
AddrToCurrent_a <= (AddrToNext_a + "111");
end if;
elsif(AddrToNext_a(3 downto 0) = "0000") then  
dataOut1(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "1000") then  
dataOut2(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "10000") then  
dataOut3(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "1011") then  
dataOut4(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "1100") then  
dataOut5(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "11000") then  
dataOut6(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "1111") then  
dataOut7(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "1110") then  
dataOut8(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "11110") then  
dataOut9(31 downto 0) <= qFromNext_a;
elsif(AddrToNext_a(3 downto 0) = "11111") then  
dataOut10(31 downto 0) <= qFromNext_a;
else  
sr_counter <= (sr_counter + 1);
end if;
sr_counter <= (sr_counter + 1);
else  
if sr_pos = 32 and currentPositionInOutput = -1 then  
beginingRow = '0' and  
currentPositionInOutput = 0 or  
currentPositionInOutput = -1) then  
wrenToCurrent_a <= '1';  
dataToCurrent_a <= outRegister;  
AddrToCurrent_a <= (AddrToNext_a + "111");  
--outRegister <= (others => '0');  
pass <= '1';  
currentPositionInOutput <= 31;  
elsif sr_pos = 31 and currentPositionInOutput = 0 then  
--pass <= '1';  
--currentPositionInOutput <= 31;
elsif sr_pos = 31 and currentPositionInOutput = 0 then  
pass <= '1';  
currentPositionInOutput <= 31;
elsif (sr_pos > 1) and pass_counter < 8 then  
if (dataOut1(sr_pos) = '1' and pass_counter = 0) then  
countNeighbors <= (countNeighbors + dataOut1(sr_pos));  
end if;
if (dataOut1(sr_pos-1) = '1' and pass_counter = 1) then  
countNeighbors <= (countNeighbors + dataOut1(sr_pos-1));  
end if;
if (dataOut1(sr_pos-2) = '1' and pass_counter = 2) then  
countNeighbors <= (countNeighbors + dataOut1(sr_pos-2));  
end if;
if (dataOut2(sr_pos) = '1' and pass_counter = 3) then  
countNeighbors <= (countNeighbors + dataOut2(sr_pos));
if dataOut2(sr_pos-2) = '1' and pass_counter = 4 then
  countNeighbors <= (countNeighbors + dataOut2(sr_pos-2));
end if;
if dataOut3(sr_pos) = '1' and pass_counter = 5 then
  countNeighbors <= (countNeighbors + dataOut3(sr_pos));
end if;
if dataOut3(sr_pos-1) = '1' and pass_counter = 6 then
  countNeighbors <= (countNeighbors + dataOut3(sr_pos-1));
end if;
if dataOut3(sr_pos-2) = '1' and pass_counter = 7 then
  countNeighbors <= (countNeighbors + dataOut3(sr_pos-2));
end if;
if pass <= '0';
  pass_counter <= (pass_counter + 1);
elsif pass = '0' then
  if dataOut2(sr_pos-1) = '1' then
    countNeighbors <= (countNeighbors + 1);
  if countNeighbors = "0010" or countNeighbors = "0011" then
    --if beginningRow = '1' then
      --outRegister(sr_pos-1) <= '1';
    --else
      outRegister(currentPositionInOutput) <= '1';
    --end if;
  else
    --if beginningRow = '1' then
      --outRegister(sr_pos-1) <= '0';
    --else
      outRegister(currentPositionInOutput) <= '0';
    --end if;
  end if;
else
  if countNeighbors = "0011" then
    --if beginningRow = '1' then
      --outRegister(sr_pos-1) <= '1';
    --else
      outRegister(currentPositionInOutput) <= '1';
    --end if;
  else
    --if beginningRow = '1' then
      --outRegister(sr_pos-1) <= '0';
    --else
      outRegister(currentPositionInOutput) <= '0';
    --end if;
  end if;
end if;
if sr_pos = 2 then
  wrenToCurrent_a <= '0';
end if;
countNeighbors <= "0000";
currentPositionInOutput <= (currentPositionInOutput - 1);
pass_counter <= "0000";
sr_pos <= (sr_pos - 1);
pass <= '1';
else
  if beginningRow = '0' then

outRegister(31) <= '0';
end if;
dataOut1(33 downto 32) <= dataOut1(1 downto 0);
dataOut2(33 downto 32) <= dataOut2(1 downto 0);
dataOut3(33 downto 32) <= dataOut3(1 downto 0);
AddrToNext_a <= AddrToNext_a + 1;
load_new <= '1';
pass <= '1';
address <= (others => '0');
pass_counter <= "0000";
beginningRow <= '0';
sr_counter <= "0000";
sr_pos <= 33;
countNeighbors <= "0000";
end if;
end if;
elsif update = '0' and done_update = '1' then
  dataOut1 <= "0000000000000000000000000000000000";
dataOut2 <= "0000000000000000000000000000000000";
dataOut3 <= "0000000000000000000000000000000000";
outRegister <= (others => '0');
address <= (others => '0');
sr_counter <= "0000";
wrenToNext_a <= '0';
wrenToCurrent_a <= '0';
AddrToNext_a <= "00000000000";
AddrToCurrent_a <= "00000000000";
done_update <= '0';
hexToHEX <= (hexToHEX + 1);
swap <= (not swap);
end if;
end if;
end process updater;

HEX <= hexToHEX;

-- Rectangle generator

RectangleHGen : process (clk25_diff, reset)
begin
  if reset = '1' then
    rectangle_h <= '1';
  elsif clk25_diff'event and clk25_diff = '1' then
    if Vcount = VSYNC + VBACK_PORCH - 1 + 111 then
      if done_update = '1' then
        update <= '0';
      end if;
      rectangle_h <= '0';
      VGAReg <= qFromCurrent_b;
      bitPos <= "11111";
    elsif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
      rectangle_h <= '0';
      AddrToCurrent_b <= "00000000000";
    end if;
  end if;
end process RectangleHGen;
displayRate <= (displayRate + 1);
if(displayRate = "00000000111111111") then
  displayRate <= "000000000000000000";
  update <= '1';
end if;
elsif (Hcount > HSYNC + HBACK_PORCH + 191) and (Hcount < HSYNC + HBACK_PORCH + 449) and
  (Vcount > VSYNC + VBACK_PORCH - 1 + 111) and (Vcount < VSYNC + VBACK_PORCH - 1 + 368) then
  videoBit <= VGAReg(conv_integer(bitPos));
  if (videoBit = '1') then
    rectangle_h <= '1';
  else
    rectangle_h <= '0';
  end if;
elsif (Hcount > HSYNC + HBACK_PORCH + 191) and (Hcount < HSYNC + HBACK_PORCH + 449) and
  (Vcount > VSYNC + VBACK_PORCH - 1 + 111) and (Vcount < VSYNC + VBACK_PORCH - 1 + 368) then
  videoBit <= VGAReg(conv_integer(bitPos));
  if (videoBit = '1') then
    rectangle_h <= '1';
  else
    rectangle_h <= '0';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elsif (Hcount > HSYNC + HBACK_PORCH + 191) and (Hcount < HSYNC + HBACK_PORCH + 449) and
  (Vcount > VSYNC + VBACK_PORCH - 1 + 111) and (Vcount < VSYNC + VBACK_PORCH - 1 + 368) then
  videoBit <= VGAReg(conv_integer(bitPos));
  if (videoBit = '1') then
    rectangle_h <= '1';
  else
    rectangle_h <= '0';
  end if;
elsif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 369) then
  rectangle_h <= '0';
  AddrToNext_b <= "00000000000";
  displayRate <= (displayRate + 1);
  if(displayRate = "00000000111111111") then
    displayRate <= "000000000000000000";
    --hexToHEX <= (hexToHEX + 1);
    update <= '1';
  end if;
elseif (Vcount = VSYNC + VBACK_PORCH - 1 + 111) then
  update <= '0';
  rectangle_h <= '0';
  VGAReg <= qFromCurrent_b;
  bitPos <= "11111";
end if;

bitPos <= (bitPos - 1);
else
  bitPos <= "11111";
  VGAReg <= qFromNext_b;
  rectangle_h <= '0';
end if;
end if;
end if;
end process RectangleHGen;

RectangleVGen : process (clk25_diff, reset)
begin
  if reset = '1' then
    rectangle_v <= '0';
  elsif clk25_diff'event and clk25_diff = '1' then
    if ram_loaded = '0' then
      rectangle_v <= '0';
    elsif EndOfLine = '1' then
      if (Vcount > VSYNC + VBACK_PORCH - 1 + 367) or (Vcount < VSYNC + VBACK_PORCH - 1 + 112) then
        rectangle_v <= '0';
      else
        rectangle_v <= '1';
      end if;
    end if;
  end if;
end process RectangleVGen;

rectangle <= rectangle_h and rectangle_v;

-- Registered video signals going to the video DAC

VideoOut: process (clk25_diff, reset)
begin
  if reset = '1' then
    VGA_R <= "0000000000";
    VGA_G <= "0000000000";
    VGA_B <= "0000000000";
  elsif clk25_diff'event and clk25_diff = '1' then
    if rectangle = '1' then
      VGA_R <= "1111111111";
      VGA_G <= "1111111111";
      VGA_B <= "1111111111";
    elsif vga_hblank = '0' and vga_vblank = '0' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "1111111111";
    else
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    end if;
  end if;
end process VideoOut;
VGA_CLK <= clk25_diff;
VGA_HS <= not vga_hsync;
VGA_VS <= not vga_vsync;
VGA_SYNC <= '0';
VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <io.h>
#include <system.h>
#include <time.h>

#define IOWR_VGA_DATA(base,offset,data) IOWR_32DIRECT(base,offset,data)
#define IORD_VGA_DATA(base,offset) IORD_32DIRECT(base,offset)

int main(int argc, char* argv[])
{
    unsigned int thisNumber=0;
    /*
    *To use each of these two blocks of if statements, make sure
    * that thisNumber = rand() (above) is commented out.
    *
    *Border-test flicker
    */

    while(i < 8192)
    {
        thisNumber = rand();

        /*
        *Glider test
        */

        if (i == 1044)
            thisNumber = 4;
        if (i == 1076)
            thisNumber = 3;

        if(i == 1044)
            thisNumber = 1;
        if(i == 1076)
            thisNumber = 1;
        if(i == 1108)
            thisNumber = 1;
    }

    printf("Seeded and preparing to write...\n");

    return 0;
}
if (i == 1108)
    thisNumber = 6;
*/

if (i < 32 || i > 8156)
    thisNumber = 0;
printf("Writing %x to: %d\n",thisNumber,i);
IOWR_VGA_DATA(VGA_BASE,i,thisNumber);
i+=4;
while(j < 1000){
    ++j;
    j=0;
}

return 0;