Game of Complex Life

Modeling of Urban Growth Processes with Cellular Automata

Participant: Nikolaos Kyparissas
Supervisor: Prof. Apostolos Dollas

Technical University of Crete
Greece

Team Number: XIL-64297

YouTube video: https://youtu.be/svI7nAp2KHE

(This page is intentionally left blank.)

Table of Contents

1. Introduction 5
2. Design 7
    2.1. Hardware 10
        2.1.1 Block Random Access Memory (BRAM) Modules 12
        2.1.2 Digital Clock Management Unit 12
        2.1.3. Debounce Units 12
        2.1.4 Cellular Automaton Module 13
        2.1.5 Graphics Controller 14
        2.1.6 Speed Controller 15
        2.1.7 7-Segment Controller 16
    2.2. Design Reuse 17
3. Results 18
4. Conclusion 19
5. References 19
6. Source Code for User Generated Hardware and Constraints 20

(This page is intentionally left blank.)

1. Introduction

With the rapid increase of cities' population number, particularly in first-world countries, calculating a city's urban growth has become an essential need for civil engineers, architects and other civil or municipal agents.

A city's growth is limited by geographical factors. It is also affected by the increasing attractiveness of certain areas of the city, like downtown or major commercial and industrial centers.

Many interesting mathematical models have been used to calculate urban growth over time, and cellular automata have been by far one of the most efficient. Cellular automata's main advantage is their flexible rules - just by setting a few simple states and rules, a cellular automaton can model incredibly complex systems.

The aim of this project was to implement a simple cellular automaton on FPGA technology which emulates such urban growth processes, based on some of the aforementioned factors.

This report presents the design and its components, as well as its operation.

This project was implemented as part of Xilinx's Open Hardware 2015 competition.

XILINX | UNIVERSITY PROGRAM
Open Hardware 2015

2. Design

The design was implemented in VHDL, using Xilinx Vivado 2014.4. Digilent's Nexys 4 Artix-7 FPGA Board, which was connected to a monitor via a VGA cable, was used for testing. Our design implements a cellular automaton, which, given a starting map of a town and its surrounding area as a grid, emulates this town's growth through time.

A cellular automaton consists of a regular grid of cells, each in one of a finite number of states. For each cell, a set of cells called its neighborhood is defined relative to the specified cell. An initial state (time t = 0) is selected by assigning a state for each cell. A new generation is created (advancing t by 1) according to some fixed mathematical rules that determine the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood.

Thus, double buffering constitutes the best choice for us - a completed timestamp of our cellular automaton's state is presented to the user, while the same timestamp's data is processed by the cellular automaton's rules logic in order to produce the next timestamp.

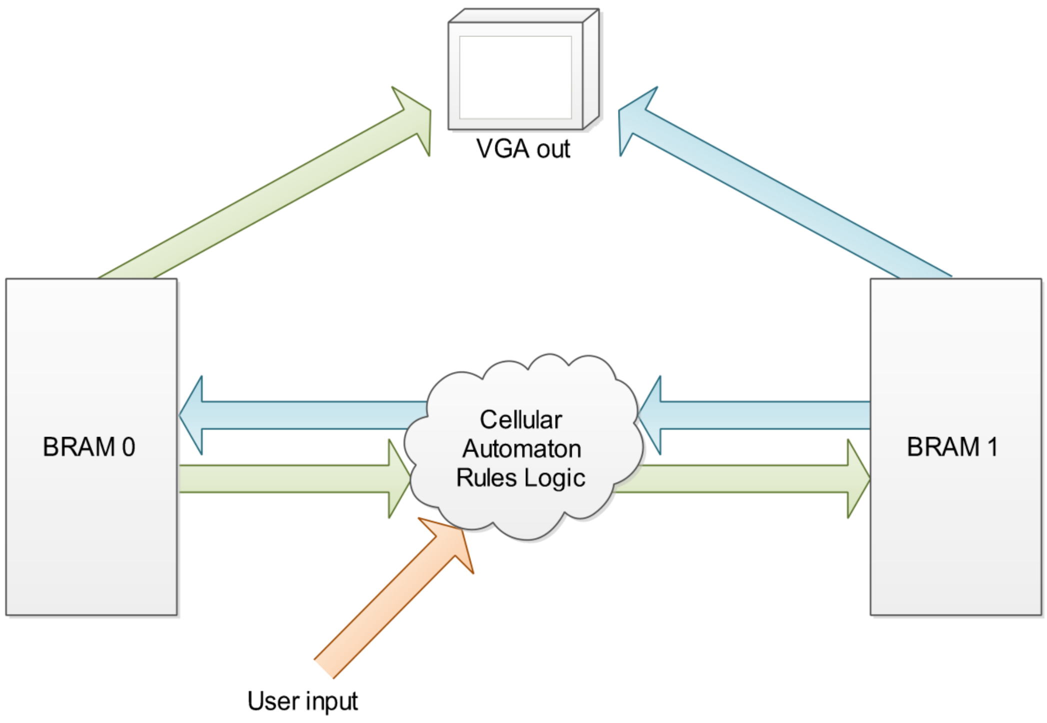


Figure 2.1: Double Buffering

Before our device begins its operation, both of its BRAM modules must be loaded with the starting state of the 640x400 grid. The cellular automaton's grid is represented in memory as shown in Figure 2.2. Every 8-bit memory slot represents a cell on the grid, and contains the cell's state. The grid's cells are stored in the memory row by row.

The same data is used by the Graphics Controller to display the grid via VGA, providing a resolution of 1280x800 pixels.

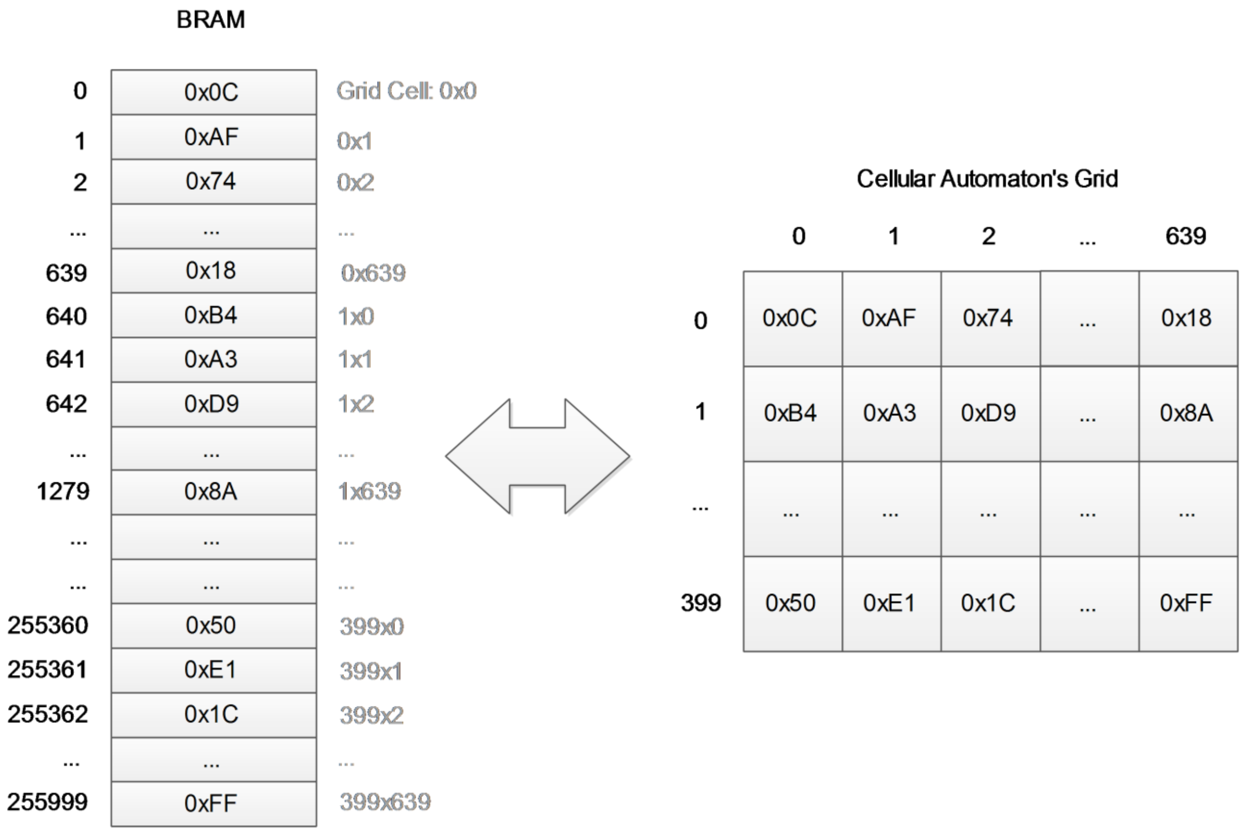


Figure 2.2: Memory representation of the cellular automaton's grid

Each Cell has 8 possible states:

1. Simple Residences / Suburbs: grey cells, 0b11011011,
2. Commercial / Employment Centers: red cells, 0b11100000,
3. Apartment Buildings / City Center: white cells, 0b11111111,
4. Parks: vivid green cells, 0b00011000,
5. Plains: dark green cells, 0b00001100,
6. Low Hills: green cells, 0b01010000,
7. High Hills: light green cells, 0b01110100,
8. Unbuildable Area: any other color value that matches the map's attributes, for example dark brown for high mountains or blue for the sea.

Below you can see an example of such a map, ready to be loaded on our design's memory.

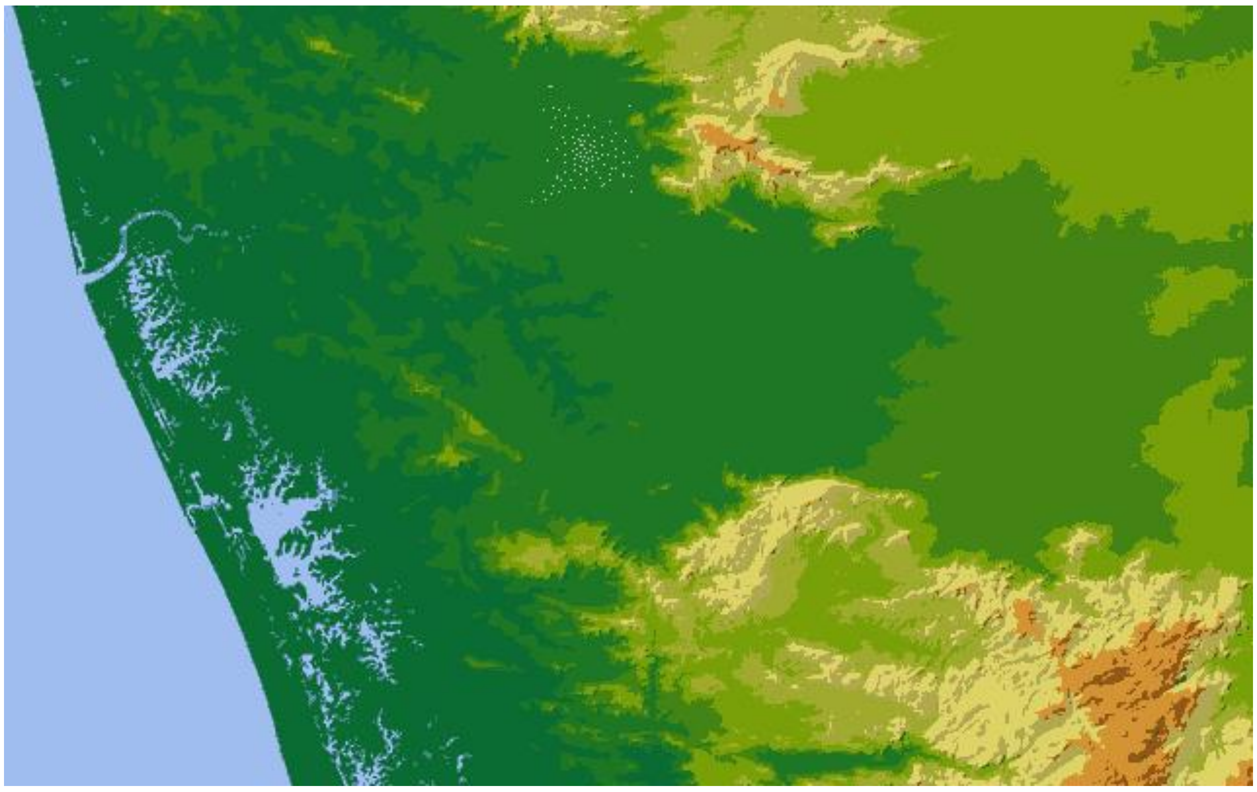


Figure 2.3: An example of a map, that fulfills the cellular automaton's rules' requirements

Our cellular automaton processes each cell serially and, according to its rules, the cell's current state and the cell's neighborhood state, it decides what the cell's next state will be.

The type of neighborhood our cellular automaton's rules process is an extended version of Moore Neighborhood. It consists of the 48 adjacent cells to the one that is currently being processed.

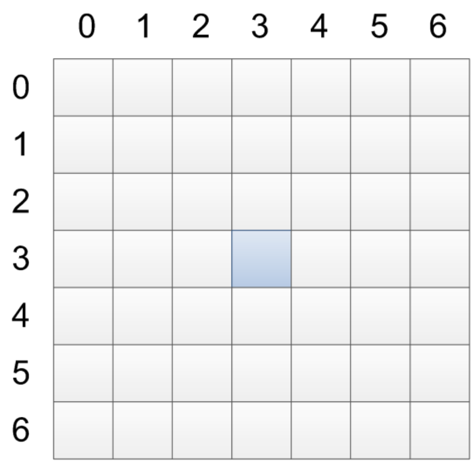


Figure 2.4: Our rules use an extended version of Moore Neighborhood

A brief description of the cellular automaton's rules follows:

1. The state of the cell we are about to process has to be either that of an urban construction or that of a buildable area.
2. If in the extended Moore Neighborhood of the cell we are about to process the number of Simple Residences and Commercial / Employment Centers is equal to four and its regular Moore Neighborhood is empty, then a Simple Residence / Suburbs Block will be constructed.
   These rules assure the existence of a pseudo-randomness in the way the suburbs grow, as far as this size of map and the scale of our emulation are concerned.
3. If the cell we are about to process belongs to a City Center's buildable area and there are more than five Simple Residents and Commercial / Employment Centers in its extended Moore Neighborhood, an Apartment Building / City Center will be built.
4. Commercial / Employment Centers' construction depends on whether they are located in the suburbs or downtown.
5. If the cell we are about to process belongs to a park area and there are enough Apartment Buildings in its extended Moore Neighborbood, then a park will be constructed there.
6. If the cell we are about to process is a Commercial / Employment Center and it belongs to a park's regular Moore Neighborhood, it will be torn down and the park will be expanded in its place.
7. If the area is not buildable, the state of the cell remains as is.

2.1 Hardware

Our design consists of seven hardware components (modules), excluding Switch Debouncer Circuits, Multiplexers and Demultiplexers.

Our design accepts three inputs:

- RESET (Nexys 4's Left Push Button),
- PAUSE/STEP (Nexys 4's Central Push Button),
- AUTO (Nexys 4's Upper Push Button),

and generates two outputs:

- The cellular automaton's current state in 1280x800 graphics via VGA,
- The design's operational status: manual or automatic via four 7-segment display digits, which display "StEP" or "AUtO" respectively.

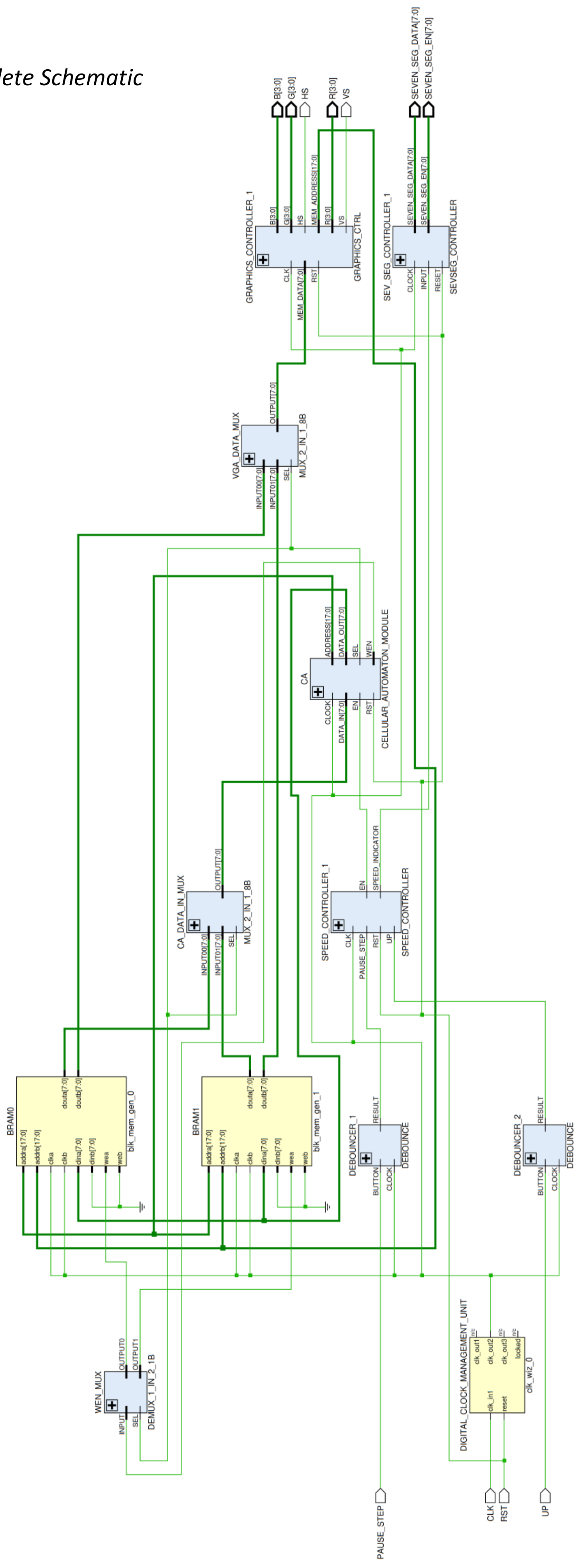
The pace at which the cellular automaton's generations are created depends on the inputs PAUSE/STEP or AUTO. PAUSE/STEP pauses the design's operation, which then awaits for PAUSE/STEP to be activated again so that it creates the cellular automaton's next generation. If AUTO is activated, our design automatically creates generations at maximum pace.

The pace selected by the user is shown to them via the 7-segment display throughout the device's operation.

When RESET input is activated, the system initializes all the internal signals and outputs.

A complete schematic of our design and brief description of each hardware component are next to follow.

Figure 2.5: Complete Schematic of our design.



2.1.1 Block Random Access Memory (BRAM) Modules

Two Dual Port BRAM modules have been used in order to implement double buffering. Both ports are writable. They were created by *Xilinx LogiCORE IP Block Memory Generator*. Both must have been loaded with the same .coe file, which contains the cellular automaton's grid's starting state.

2.1.2 Digital Clock Management Unit

The system's Digital Clock Management Unit receives a 100 MHz external clock signal and outputs an 83.46 MHz clock signal, suitable for our Graphics Controller's WXGA (1280x800@60Hz) output.

This module was created using *Xilinx Clocking Wizard*.

2.1.3. Debounce Units

Debounce units help stabilize any input signals that may flicker due to the board's push buttons' structure.

They introduce a two-level FIFO buffer which has its values compared to the input at a pace relevant to the system's clock frequency.

These modules' design was *based on example designs available* at *www.eewiki.net* .

2.1.4 Cellular Automaton Module

The Cellular Automaton Module was completely designed by the participant. It is responsible for processing the cellular automaton. After loading the cell that is about to be processed along with its 7x7 extended Moore Neighborhood from the memory that represents the previously completed timestamp of our cellular automaton, it then calculates its new value and writes it back to the memory module that currently represents the next timestamp. It then loads and processes the next cell of the grid.

When it has completed processing the last cell of the grid, it starts working on the next timestamp from the beginning, using the data it just stored by setting its SEL output accordingly, and effectively interchanging the memory modules' roles.

In more details, it implements the following Finite State Machine:

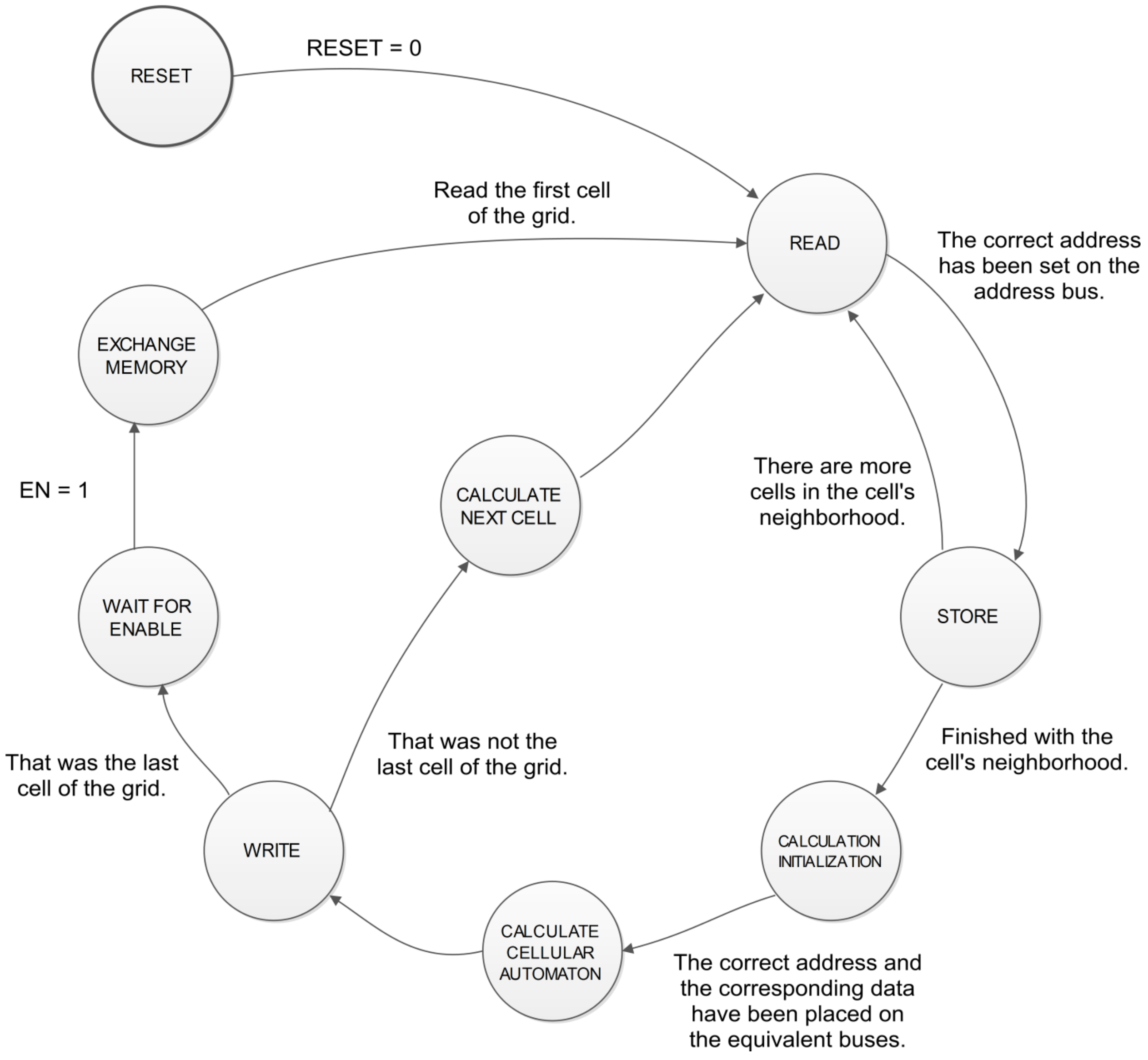

RESET
RESET = 0
Read the first cell of the grid.
READ
The correct address has been set on the address bus.
EXCHANGE MEMORY
EN = 1
CALCULATE NEXT CELL
There are more cells in the cell's neighborhood.
STORE
WAIT FOR ENABLE
That was not the last cell of the grid.
Finished with the cell's neighborhood.
That was the last cell of the grid.
WRITE
CALCULATION INITIALIZATION
CALCULATE CELLULAR AUTOMATON
The correct address and the corresponding data have been placed on the equivalent buses.

Figure 2.6: The Finite State Machine implemented by the Cellular Automaton Module

2.1.5 Graphics Controller

The Graphics Controller consists of two internal hardware components, WXGA Sync Controller and WXGA Color Controller.

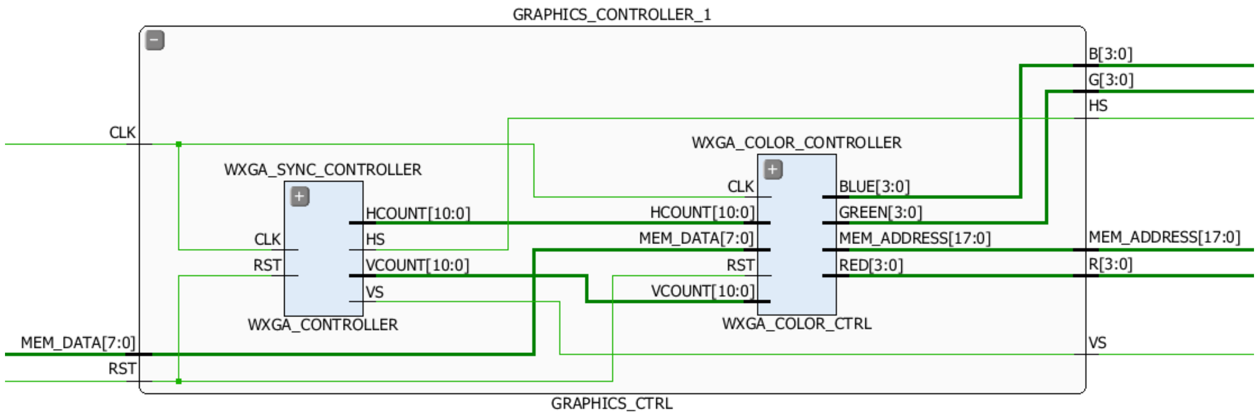


Figure 2.7: The Graphics Controller, its components and their interconnections

WXGA Sync Controller is *based on Ulirch Zoltan's "VGA Controller" (Digilent, 2006)*. This module generates the video synch pulses for the monitor to enter 1280x800@60hz resolution state (WXGA). It also provides horizontal and vertical counters for the currently displayed pixel and a blank signal that is active when the pixel is not inside the visible screen.

WXGA Color Controller was completely designed by the participant. It reads serially 8-bit data from an external memory, transforms it into a 12-bit color output signal, and transmits it according to WXGA Sync Controller's instructions (Horizontal Counter and Vertical Counter signals).

2.1.6 Speed Controller

The Speed Controller module was completely designed by the participant. It either starts generating an Enable signal constantly once its UP input becomes activated, or whenever its PAUSE/STEP input becomes activated.

It also indicates the way it is producing the Enable signal, through an output called Speed Indicator.

In more details, it implements the following Finite State Machine:

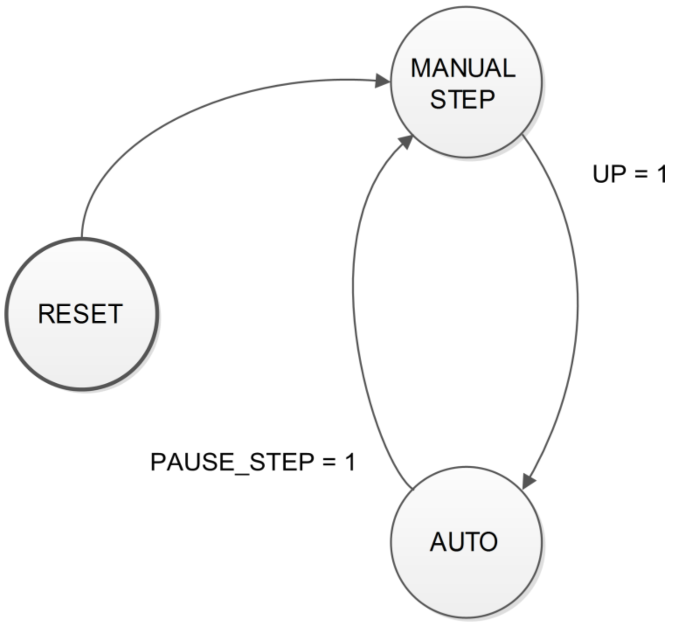


Figure 2.8: The Finite State Machine implemented by the Speed Controller module

2.1.7 7-Segment Controller

The 7-Segment Controller module was completely designed by the participant. It multiplexes four different 7-segment digits of a 7-segment display. Implementing a Finite State Machine which lights up (enables) every digit for about 4 msec, it displays "StEP" or "AUtO", depending on its input.

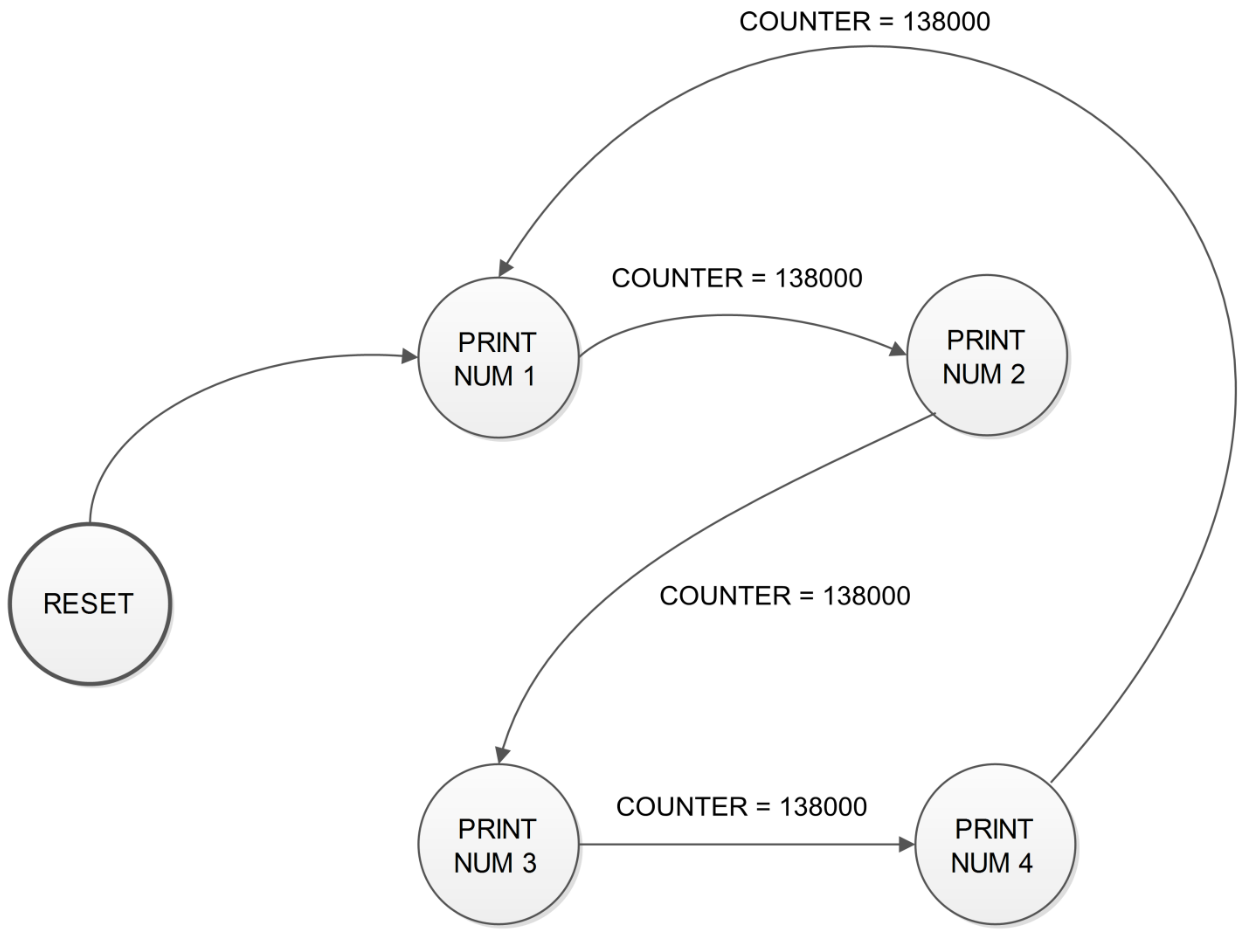


Figure 2.9: The Finite State Machine implemented by the 7-Segment Controller module

2.2 Design Reuse

This design will emulate an urban growth process given any map data that fulfills the cellular automaton's rules' requirements. A map's content can vary from a small town which will gradually develop, to a few large cities which will eventually merge and create a vast metropolis.

Additionally, a generic code of the Cellular Automaton Module has been uploaded on the participant's github profile (https://github.com/nkyparissas/VHDL → Open Hardware 2015). Any designer who wants to implement their own cellular automaton, can adjust the grid and neighborhood size generic variables, and apply their own rules without losing precious time dealing with addressing calculations.

Using this code the designer can create a cellular automaton of varying complexity due to the ability to adjust the size of a cell's neighborhood. Large neighborhoods combined with complex rules can result in realistic emulations or even simulations of large-scale natural processes.

3. Results

The main challenges implementing the design were the complex addressing of the cellular automaton processing and achieving realistic modeling of urban growth processes, based on basic factors due to several resources' restrictions.

The result is a realistic emulation of basic urban growth processes, which takes into account geographical and land-planning factors.

The bottleneck was the system clock's frequency, which had to be the same with the WXGA clock frequency, at 83.46 MHz, and the amount of BRAM available, both being problems that can be solved in the future (see below, in the "Conclusion" chapter).

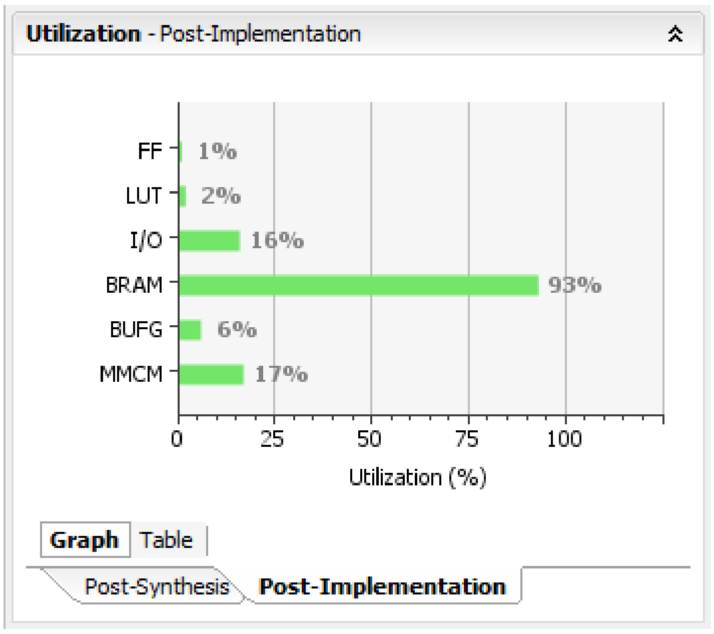


Figure 3: Resource Utilization

With more memory available, other factors could have been taken into account as far as the cellular automaton's rules are concerned, like the cell's age, attractiveness, accessibility etc.

The design processes 205416 cells in order to create a new generation, and that results in 9859968 memory accesses per generation (205416 x 48, the cell's neighborhood).

You can watch Game of Complex Life in action, if you follow the following link:

Game of Complex Life - Xilinx XUP OpenHardware2015 + XIL-64297:
https://youtu.be/svI7nAp2KHE

4. Conclusion

Game of Complex Life emulates basic urban growth processes successfully, taking into account geographical and land-planning factors, and producing interesting results. FPGA technology allows us to take full advantage of interesting and efficient hardware design methods like double buffering.

The problems we encountered include the complex addressing needed for processing, and several resources restrictions.

In the future we want to take advantage of the additional options *Xilinx Clocking Wizard* gives us in order for the Cellular Automaton Module to be able to operate in even higher frequencies. In addition, we will use the board's external RAM in order to create more complex rules that take advantage of factors like age, attractiveness, accessibility, zoning, distance from the original areas etc.

This will give us the ability to realistically simulate the urban growth of real cities through time.

5. References

von Neumann, J. (1948). *The general and logical theory of automata*, in L.A. Jeffress, ed., *Cerebral Mechanisms in Behavior – The Hixon Symposium*, John Wiley & Sons, New York, 1951, pp. 1–31.

von Neumann, J.; Burks, A. W. (1966). *Theory of Self-Reproducing Automata*, University of Illinois Press.

Iannone, G.; Troisi, A.; Guarnaccia, C.; D'Agostino, P. P.; Quartieri, J. (2011) *'An Urban Growth Model Based on a Cellular Automata Phenomenological Framework',* International Journal of Modern Physics C, Vol. 22(No. 5), pp. 543.

Shlomo, A.; Parent, J.; Civco, D. L.; Blei, A. M. (2011) *Making Room for a Planet of Cities*, Cambridge, MA, USA: Lincoln Institute of Land Policy.

6. Source Code for User Generated Hardware and Constraints

```
----------------------------------------------------------------------------------
    -- TECHNICAL UNIVERSITY OF CRETE
    -- NICK KYPARISSAS
    --
    -- CREATE DATE: 16 JUNE 2015
    -- MODULE NAME: GAME_OF_COMPLEX_LIFE - BEHAVIORAL
    -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
    --
    -- DESCRIPTION: TOP LEVEL MODULE.
----------------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

LIBRARY UNISIM;
USE UNISIM.VCOMPONENTS.ALL;

ENTITY GAME_OF_COMPLEX_LIFE IS
    PORT (
        CLK             : IN STD_LOGIC; -- @ 83.46 MHZ
        RST             : IN STD_LOGIC;
        UP              : IN STD_LOGIC;
        PAUSE_STEP      : IN STD_LOGIC;
        -- 7 SEGMENT DISPLAY
        SEVEN_SEG_EN   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        SEVEN_SEG_DATA  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        -- VGA
        HS              : OUT STD_LOGIC;
        VS              : OUT STD_LOGIC;
        R               : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        G               : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        B               : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END GAME_OF_COMPLEX_LIFE;

ARCHITECTURE BEHAVIORAL OF GAME_OF_COMPLEX_LIFE IS
    --SIGNALS
    SIGNAL VGA_ADDR_SIG                 : STD_LOGIC_VECTOR(17 DOWNTO 0) := (OTHERS => '0');
    SIGNAL VGA_DATA_SIG                 : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL VGA_DATA_0_SIG               : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL VGA_DATA_1_SIG               : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL ADDR_SIG                     : STD_LOGIC_VECTOR(17 DOWNTO 0) := (OTHERS => '0');
    SIGNAL DATA_OUT_SIG                 : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL DATA_IN_0_SIG                : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL DATA_IN_1_SIG                : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL DATA_IN_SIG                  : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL WEN_SIG                      : STD_LOGIC_VECTOR(0 DOWNTO 0) := (OTHERS => '0');
    SIGNAL WEA_0_SIG                    : STD_LOGIC_VECTOR(0 DOWNTO 0) := (OTHERS => '0');
    SIGNAL WEA_1_SIG                    : STD_LOGIC_VECTOR(0 DOWNTO 0) := (OTHERS => '0');
    SIGNAL SPEED_INDICATOR_SIG          : STD_LOGIC;
    SIGNAL DEBOUNCED_STEP_SIG, DEBOUNCED_UP_SIG : STD_LOGIC;
    SIGNAL SEL_SIG, CLK_SIG, EN_SIG     : STD_LOGIC;

    SIGNAL DEBOUNCED_EN_SIG             : STD_LOGIC;
    --COMPONENTS
    COMPONENT GRAPHICS_CTRL
        PORT (
            CLK      : IN STD_LOGIC;  -- @ 83.46 MHZ
            RST      : IN STD_LOGIC;
            MEM_DATA     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            HS          : OUT STD_LOGIC;
            VS          : OUT STD_LOGIC;
            R           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            G           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            B           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            MEM_ADDRESS : OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT CELLULAR_AUTOMATON_MODULE
        PORT (
            DATA_IN          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            EN               : IN STD_LOGIC;
            RST              : IN STD_LOGIC;
            CLOCK            : IN STD_LOGIC;
            WEN              : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
            SEL              : OUT STD_LOGIC;
            ADDRESS          : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
        DATA_OUT         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
    END COMPONENT;

    COMPONENT SPEED_CONTROLLER
        PORT (
            UP : IN STD_LOGIC;
            PAUSE_STEP : IN STD_LOGIC;
            CLK : IN STD_LOGIC;
            RST : IN STD_LOGIC;
            EN : OUT STD_LOGIC;
        SPEED_INDICATOR : OUT STD_LOGIC);
    END COMPONENT;
```

```
 COMPONENT SEVSEG_CONTROLLER
     PORT (
         CLOCK : IN  STD_LOGIC;
         RESET : IN  STD_LOGIC;
         INPUT : IN STD_LOGIC;
         SEVEN_SEG_EN : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
     SEVEN_SEG_DATA : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

 COMPONENT CLK_WIZ_0
     PORT(-- CLOCK IN PORTS
         CLK_IN1           : IN     STD_LOGIC;
         -- CLOCK OUT PORTS
         CLK_OUT1          : OUT    STD_LOGIC;
         CLK_OUT2          : OUT    STD_LOGIC;
         CLK_OUT3          : OUT    STD_LOGIC;
         -- STATUS AND CONTROL SIGNALS
         RESET             : IN     STD_LOGIC;
         LOCKED            : OUT    STD_LOGIC
     );
END COMPONENT;

 COMPONENT BLK_MEM_GEN_0
     PORT (
         CLKA : IN STD_LOGIC;
         WEA : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         ADDRA : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
         DINA : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         DOUTA : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
         CLKB : IN STD_LOGIC;
         WEB : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         ADDRB : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
         DINB : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         DOUTB : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
     );
END COMPONENT;

 COMPONENT BLK_MEM_GEN_1
     PORT (
         CLKA : IN STD_LOGIC;
         WEA : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         ADDRA : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
         DINA : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         DOUTA : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
         CLKB : IN STD_LOGIC;
         WEB : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
         ADDRB : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
         DINB : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         DOUTB : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
     );
END COMPONENT;

 COMPONENT DEBOUNCE
     PORT(
         CLOCK   : IN  STD_LOGIC;  --INPUT CLOCK.
         BUTTON  : IN  STD_LOGIC;  --INPUT SIGNAL TO BE DEBOUNCED.
         RESULT  : OUT STD_LOGIC --DEBOUNCED SIGNAL.
     );
END COMPONENT;

 COMPONENT MUX_2_IN_1_8B
     PORT (
         INPUT00  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         INPUT01  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         SEL      : IN STD_LOGIC;
     OUTPUT   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END COMPONENT;

 COMPONENT DEMUX_1_IN_2_1B
     PORT (
         INPUT    : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
         SEL      : IN STD_LOGIC;
         OUTPUT0  : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
     OUTPUT1  : OUT STD_LOGIC_VECTOR (0 DOWNTO 0));
END COMPONENT;

BEGIN

 CA: CELLULAR_AUTOMATON_MODULE
PORT MAP (
    DATA_IN => DATA_IN_SIG,
    EN => EN_SIG,
    RST => RST,
    CLOCK => CLK_SIG,
    WEN => WEN_SIG,
    SEL => SEL_SIG,
    ADDRESS => ADDR_SIG,
    DATA_OUT => DATA_OUT_SIG
);
```

```
 BRAM0: BLK_MEM_GEN_0
PORT MAP(
    CLKA => CLK_SIG,
    WEA => WEA_0_SIG,
    ADDRA => ADDR_SIG,
    DINA => DATA_OUT_SIG,
    DOUTA => DATA_IN_0_SIG,
    CLKB => CLK_SIG,
    WEB => "0",
    ADDRB => VGA_ADDR_SIG,
    DINB => "00000000",
    DOUTB => VGA_DATA_0_SIG
);

 BRAM1: BLK_MEM_GEN_1
PORT MAP(
    CLKA => CLK_SIG,
    WEA => WEA_1_SIG,
    ADDRA => ADDR_SIG,
    DINA => DATA_OUT_SIG,
    DOUTA => DATA_IN_1_SIG,
    CLKB => CLK_SIG,
    WEB => "0",
    ADDRB => VGA_ADDR_SIG,
    DINB => "00000000",
    DOUTB => VGA_DATA_1_SIG
);

 DIGITAL_CLOCK_MANAGEMENT_UNIT: CLK_WIZ_0
 PORT MAP(
     CLK_IN1 => CLK,
     CLK_OUT2 => CLK_SIG,
     RESET => RST
);

 GRAPHICS_CONTROLLER_1: GRAPHICS_CTRL
 PORT MAP(
     CLK => CLK_SIG,
     RST => RST,
     MEM_DATA => VGA_DATA_SIG,
     MEM_ADDRESS => VGA_ADDR_SIG,
     HS => HS,
     VS => VS,
     R => R,
     G => G,
     B => B
);

 SPEED_CONTROLLER_1: SPEED_CONTROLLER
PORT MAP(
    UP => DEBOUNCED_UP_SIG,
    PAUSE_STEP => DEBOUNCED_STEP_SIG,
    CLK => CLK_SIG,
    RST => RST,
    EN => EN_SIG,
SPEED_INDICATOR => SPEED_INDICATOR_SIG);


SEV_SEG_CONTROLLER_1: SEVSEG_CONTROLLER
PORT MAP(
    CLOCK => CLK_SIG,
    RESET => RST,
    INPUT => SPEED_INDICATOR_SIG,
    SEVEN_SEG_EN => SEVEN_SEG_EN,
SEVEN_SEG_DATA => SEVEN_SEG_DATA);


 VGA_DATA_MUX: MUX_2_IN_1_8B
 PORT MAP(
     INPUT00 => VGA_DATA_0_SIG,
     INPUT01 => VGA_DATA_1_SIG,
     SEL => SEL_SIG,
     OUTPUT => VGA_DATA_SIG
);

 CA_DATA_IN_MUX: MUX_2_IN_1_8B
 PORT MAP(
    INPUT00 => DATA_IN_0_SIG,
    INPUT01 => DATA_IN_1_SIG,
    SEL => SEL_SIG,
    OUTPUT => DATA_IN_SIG
);

 WEN_MUX: DEMUX_1_IN_2_1B
 PORT MAP(
     INPUT => WEN_SIG,
     SEL => SEL_SIG,
     OUTPUT0 => WEA_0_SIG,
     OUTPUT1 => WEA_1_SIG
);

 DEBOUNCER_2: DEBOUNCE
 PORT MAP(
     BUTTON => UP,
     RESULT => DEBOUNCED_UP_SIG,
     CLOCK => CLK_SIG
);
```

```
    DEBOUNCER_1: DEBOUNCE
    PORT MAP(
         BUTTON => PAUSE_STEP,
         RESULT => DEBOUNCED_STEP_SIG,
         CLOCK => CLK_SIG
    );

END BEHAVIORAL;
```

```vhdl
-----------------------------------------------------------------------------------
    -- TECHNICAL UNIVERSITY OF CRETE
    -- NICK KYPARISSAS
    --
    -- CREATE DATE: 16 JUNE 2015
    -- MODULE NAME: CELLULAR_AUTOMATON_MODULE - BEHAVIORAL
    -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
    --
    -- DESCRIPTION: THIS MODULE LOADS AN EXANDING CELLULAR AUTOMATON (CA) FROM MEMORY MODULE NO. 0,
    -- ACTS ON IT ACCORDING TO THE CA'S RULES AND STORES THE RESULT IN MEMORY MODULE NO. 1.
    -- THEN IT EXCHANGES THE MEMORY MODULES, READS FROM MEMORY MODULE NO. 1 AND WRITES IN MEMORY MODULE NO. 0.
    -- DUE TO LIMITED MEMORY SIZE, WE HAD TO AVOID HAVING DYING CELLS (BECAUSE THEN YOU NEED MORE MEMORY TO
    -- STORE A 3RD FRAME WITHOUT ANY CELLS ON IT), SO THIS MODULE IMPLEMENTS A FOREVER EXPANDING CA.
    -- IMPLEMENTING A DYING CELL WOULD BE EASY: WE JUST NEED 2 MORE FSM STATES AND A NEVER-CHANGING FRAME OF THE
    -- MAP'S INITIAL STATE STORED SOMEWHERE.
    --
    -- YOU CAN FIND A GENERIC MODULE OF THIS, ON github.com/nkyparissas/VHDL/blob/master/Open%20Hardware%202015
    -- AND APPLY YOUR RULES, IN ORDER TO CREATE YOUR OWN CA.
    -- YOU CAN CHOOSE THE SIZE OF THE GRID, THE SIZE OF YOUR CA'S MOORE NEIGHBORHOOD, THE NUMBER OF DIFFERENT
    -- TYPES OF CELLS YOU HAVE ETC, JUST BY SETTING SOME GENERIC VARIABLES.
-----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY CELLULAR_AUTOMATON_MODULE IS
    PORT ( DATA_IN          : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- BRAM DATA - EACH BRAM DATA CELL IS A CELL ON OUR GRID.
        EN                : IN STD_LOGIC;
        RST               : IN STD_LOGIC;
        CLOCK             : IN STD_LOGIC;
        WEN               : OUT STD_LOGIC_VECTOR (0 DOWNTO 0); -- WRITE ENABLE.
        SEL               : OUT STD_LOGIC; -- SEL SIGNAL / BRAM SELECT.
        ADDRESS           : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
        DATA_OUT          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END CELLULAR_AUTOMATON_MODULE;

ARCHITECTURE BEHAVIORAL OF CELLULAR_AUTOMATON_MODULE IS
    TYPE STATE IS (RESET, READ, STORE, CALCULATE_CA_INITIALIZATION, CALCULATE_CA, WRITE, CALCULATE_NEXT_CELL, WAIT_EN, EX_MEM);
    SIGNAL Y : STATE;
    SIGNAL CURRENT_CELL_X : STD_LOGIC_VECTOR(9 DOWNTO 0); -- THE CURRENT CELL'S X COORDINATE ON THE GRID.
    SIGNAL CURRENT_CELL_Y : STD_LOGIC_VECTOR(17 DOWNTO 0); -- THE CURRENT CELL'S Y COORDINATE ON THE GRID.
    SIGNAL GRID_X : STD_LOGIC_VECTOR(9 DOWNTO 0); -- THE CELL WE ARE CURRENTLY NEEDED - ONE OF THE CURRENT CELL'S NEIGHBORHOOD CELLS.
    SIGNAL GRID_Y : STD_LOGIC_VECTOR(17 DOWNTO 0);
-- WE HAVE FOUR KINDS OF ALIVE STATES - WHITE: CITY CENTER, GREY: SUBURBS, RED: COMMERCIAL AND EMPLOYMENT BLOCK, GREEN: PARKS.
    SIGNAL WHITE_COUNTER, GREY_COUNTER, RED_COUNTER, GREEN_COUNTER : STD_LOGIC_VECTOR(5 DOWNTO 0);
-- CENTER_BUILT_FLAG: DO WE HAVE A CENTER BUILT? IF THERE IS A CENTER INITIALLY, ANOTHER ONE WILL BE BUILT SOMEWHERE ELSE.
-- MOORE_NEIGHBORHOOD_EMPTY: IF OUR CURRENT CELL'S MOORE NEIGHBORHOOD (TYPICAL 3X3 AND NOT OUR EXTENDED NEIGHBORHOOD) IS EMPTY.
    SIGNAL CENTER_BUILT_FLAG, MOORE_NEIGHBORHOOD_EMPTY, CENTER_MOORE_NEIGHBORHOOD, PARK_MOORE_NEIGHBORHOOD: STD_LOGIC;
    SIGNAL SEL_SIG : STD_LOGIC; -- SEL SIG: WHICH MEMORY WE ARE READING FROM. (NOT SEL_SIG) WRITING.

BEGIN

    CA_RULES: PROCESS
        BEGIN

        WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';

        IF ( RST = '1' ) THEN
            Y <= RESET;
        ELSE
            CASE Y IS
                WHEN RESET => --WHEN RESET: INITIALIZE SIGNALS AND OUTPUTS.
                     -- OUTPUTS:
                     SEL_SIG <= '0'; -- SELECT THE FIRST BRAM.
                     WEN <= "0"; -- DO NOT WRITE ANYTHING.
                     -- THE OTHER OUTPUTS' VALUES ARE IRRELEVANT HERE.
                     -- SIGNALS:
                     CURRENT_CELL_X <= "0000000011"; -- CELL 3 (4TH CELL), 3 X 3 THE FIRST CELL ON THE GRID WITH A 7X7 EXTENDED MOORE NEIGHBORHOOD.
                     CURRENT_CELL_Y <= "000000011110000000"; -- IN ROW 3 (3*640=1920) (4TH ROW).
                     GRID_X <= (OTHERS => '0'); -- INITIATE ADDRESS 0, FIRST CELL OF THE GRID - FIRST CELL OF THE FIRST 7X7 NEIGHBORHOOD.
                     GRID_Y <= (OTHERS => '0');
                     WHITE_COUNTER <= (OTHERS => '0'); -- WE HAVEN'T COUNTED ANY WHITE CELLS IN THE EXTENDED NEIGHBORHOOD YET.
                     GREY_COUNTER <= (OTHERS => '0'); -- WE HAVEN'T COUNTED ANY GREY CELLS IN THE EXTENDED NEIGHBORHOOD YET.
                     RED_COUNTER <= (OTHERS => '0'); -- WE HAVEN'T COUNTED ANY RED CELLS IN THE EXTENDED NEIGHBORHOOD YET.
                     GREEN_COUNTER <= (OTHERS => '0'); -- WE HAVEN'T COUNTED ANY GREEN CELLS IN THE EXTENDED NEIGHBORHOOD YET.
                     CENTER_MOORE_NEIGHBORHOOD <= '0'; -- THERE IS NOTHING BUILT IN THE 3X3 NEIGHBORHOOD OF OUR CURRENT WHITE CELL.
                     PARK_MOORE_NEIGHBORHOOD <= '0'; -- THERE IS NOTHING BUILT IN THE 3X3 NEIGHBORHOOD OF OUR CURRENT GREEN CELL.
                     MOORE_NEIGHBORHOOD_EMPTY <= '1'; -- 1 = TRUE, 0 = FALSE.
                     CENTER_BUILT_FLAG <= '0'; -- 1 = TRUE, 0 = FALSE.
                     -- NEXT STATE:
                     Y <= READ;
                WHEN STORE =>
                     IF (GRID_X = CURRENT_CELL_X + 3) AND (GRID_Y = CURRENT_CELL_Y + 1920) THEN --IF THE FINAL CELL OF THE 7X7 NEIGHBORHOOD.
                         GRID_X <= CURRENT_CELL_X;
                         GRID_Y <= CURRENT_CELL_Y;
                         Y <= CALCULATE_CA_INITIALIZATION; -- GOTO CALCULATE THE CURRENT CELL'S FUTURE STATE BASED ON THE CA'S RULES.
                     ELSE -- CALCULATE NEXT ADDRESS OF THE 7X7 NEIGHBORHOOD.
                         IF (GRID_X = CURRENT_CELL_X + 3) THEN -- IF WE ARE IN THIS ROW'S FINAL CELL,
                             GRID_X <= CURRENT_CELL_X - 3; -- GOTO THE FIRST CELL OF THE NEXT ROW OF THE NEIGHBORHOOD,
                             GRID_Y <= GRID_Y + 640; -- AND CHANGE LINE.
                         ELSE
                             GRID_X <= GRID_X + 1;
                         END IF;
                         Y <= READ;
                     END IF;
                     -- CHECK THE MOORE NEIGHBORHOOD (NOT THE EXTENDED 7X7 ONE, BUT THE NORMAL 3X3 NEIGHBORHOOD).
```

```
     IF (GRID_Y > CURRENT_CELL_Y - 1280) AND (GRID_Y < CURRENT_CELL_Y + 1280) AND (GRID_X > CURRENT_CELL_X - 2)
     AND (GRID_X < CURRENT_CELL_X + 2) AND (DATA_IN = "11011011" OR DATA_IN = "11111111" OR DATA_IN = "11100000"
     OR DATA_IN = "00011000") THEN -- IF WE ARE READING ONE OF THE MOORE NEIGHBORHOOD CELLS, AND THEY ARE IN ANY CA'S STATE,
         MOORE_NEIGHBORHOOD_EMPTY <= '0'; -- THE MOORE NEIGHBORHOOD IS NOT EMPTY.
         IF (DATA_IN = "11111111") THEN
             CENTER_MOORE_NEIGHBORHOOD <= '1';
         ELSIF (DATA_IN = "00011000") THEN
             PARK_MOORE_NEIGHBORHOOD <= '1';
         END IF;
     END IF;
     -- STORE THE APPROPRIATE COUNTER VALUES
     IF (GRID_X /= CURRENT_CELL_X) AND (GRID_Y /= CURRENT_CELL_Y) THEN -- IF WE ARE NOT READING THE CURRENT CELL,
         IF (DATA_IN = "11111111") THEN -- IF WHITE,
         WHITE_COUNTER <= WHITE_COUNTER + 1; -- STORE THE APPROPRIATE VALUE.
         ELSIF (DATA_IN = "11011011") THEN -- IF GREY,
         GREY_COUNTER <= GREY_COUNTER + 1; -- STORE THE APPROPRIATE VALUE.
         ELSIF (DATA_IN = "11100000") THEN -- IF RED,
         RED_COUNTER <= RED_COUNTER + 1; -- STORE THE APPROPRIATE VALUE.
         ELSIF (DATA_IN = "00011000") THEN -- IF GREEN,
             GREEN_COUNTER <= GREEN_COUNTER + 1; -- STORE THE APPROPRIATE VALUE.
         END IF;
     END IF;
WHEN READ => -- IN "READ" STATE, THE CORRECT ADDRESS HAS BEEN SET ON THE ADDRESS BUS. IN THE NEXT STATE, BRAM WILL HAVE PLACED THE
             CORRESPONDIG DATA ON THE DATA BUS.
     Y <= STORE;
WHEN CALCULATE_CA_INITIALIZATION => -- IN "CALCULATE_CA_INITIALIZATION" STATE, THE CORRECT ADDRESS HAS BEEN SET ON THE ADDRESS BUS.
                                       IN THE NEXT STATE, BRAM WILL HAVE PLACED THE CORRESPONDIG DATA ON THE DATA BUS, AND WE CAN
                                       CALCULATE WHAT THE CURRENT CELL'S NEXT STATE WILL BE.
     Y <= CALCULATE_CA;
WHEN CALCULATE_CA =>
     IF (DATA_IN = "00001100") OR (DATA_IN = "01010000") OR (DATA_IN = "01110100") OR (DATA_IN = "11111111") OR
     (DATA_IN = "11011011") OR (DATA_IN = "11100000") OR (DATA_IN = "00011000") THEN -- IF THE CURRENT CELL IS A BUILDING, OR A
     BUILDABLE AREA OF THE MAP,
         --- PARKS
         IF (DATA_IN = "11100000") AND (PARK_MOORE_NEIGHBORHOOD = '1') THEN -- IF THE CURRENT CELL IS A COMMERCIAL CENTER, AND IT
         BELONGS IN A PARK'S MOORE NEIGHBOR HOOD,
             DATA_OUT <= "00011000"; -- TEAR IT DOWN AND EXPAND THE PARK.
         ELSIF ((DATA_IN = "11011011") OR (DATA_IN = "11111111") OR (DATA_IN = "11100000"))
         AND (GREY_COUNTER + RED_COUNTER + WHITE_COUNTER > 20) AND (GRID_X(4 DOWNTO 0) < 3) AND ((GRID_Y(11 DOWNTO 0) = 0)
         OR (GRID_Y(11 DOWNTO 0) = "010100000000") OR (GRID_Y(11 DOWNTO 0) = "001010000000")) THEN -- EQUAL TO GRID_X MOD 32 < 3 AND
         (GRID_Y/640) MOD 32 < 3, PARK AREAS, IF THERE ARE ENOUGH CENTRAL CITY'S BUILDINGS AROUND.
             DATA_OUT <= "00011000";
         --COMMERCIAL CENTERS
         ELSIF ((DATA_IN = "11011011") AND (RED_COUNTER < 1)) OR ((DATA_IN = "11111111") AND (PARK_MOORE_NEIGHBORHOOD = '0') AND
         (WHITE_COUNTER > 4) AND (RED_COUNTER  < 3)) THEN -- DIFFERENT RULES FOR COMMERCIAL CENTERS IN SUBURBS AND IN CITY CENTERS.
             DATA_OUT <= "11100000"; -- BUILD COMMERCIAL CENTER.
         --CITY
         ELSIF (GREY_COUNTER + RED_COUNTER > 5) AND (DATA_IN /= "11011011") AND (DATA_IN /= "11100000") AND (DATA_IN /= "00011000")
         AND (DATA_IN /= "01110100") THEN -- BUILD CITY RULES.
             IF (CENTER_BUILT_FLAG = '0') THEN
                 DATA_OUT <= "11111111";
                 CENTER_BUILT_FLAG <= '1';
             ELSIF (CENTER_MOORE_NEIGHBORHOOD = '1') AND (SEL_SIG = '0') THEN -- CITY GROWS EVERY 2 FRAMES.
                 DATA_OUT <= "11111111";
             ELSE
                 DATA_OUT <= DATA_IN;
             END IF;
         --SUBURBS
         ELSIF (GREY_COUNTER + RED_COUNTER = 4) AND (MOORE_NEIGHBORHOOD_EMPTY = '1') THEN --ELSIF (WHITE_COUNTER + GREY_COUNTER +
         RED_COUNTER = 4) AND (MOORE_NEIGHBORHOOD_EMPTY = '1') THEN -- BUILD SUBURBS (GREY)
             DATA_OUT <= "11011011";
         ELSE
             DATA_OUT <= DATA_IN;
         END IF;
     ELSE -- IF NONE OF THE RULES APPLY, THEN LEAVE THE CURRENT CELL'S STATE AS IS.
         DATA_OUT <= DATA_IN;
     END IF;
     WEN <= "1"; -- WRITE IN THE NEXT STATE.
     Y <= WRITE;
WHEN WRITE =>
    IF (CURRENT_CELL_X = 636 AND CURRENT_CELL_Y = 253440) THEN -- IF WE ARE IN THE FINAL CELL OF THE GRID, X = 636, Y = 253440 =
                                                                  (396*640).
         Y <= WAIT_EN;
    ELSE
        Y <= CALCULATE_NEXT_CELL;
     END IF;
    WEN <= "0"; -- STOP WRITING.
WHEN CALCULATE_NEXT_CELL =>
    IF (CURRENT_CELL_X = 636) THEN
        CURRENT_CELL_X <= "0000000011";
        CURRENT_CELL_Y <= CURRENT_CELL_Y + 640; -- CHANGE LINE.
        GRID_X <= (OTHERS => '0'); -- INITIATE ADDRESS POINTING AT THE FIRST CELL OF THE NEXT 7X7 NEIGHBORHOOD.
         GRID_Y <= CURRENT_CELL_Y - 1280;
    ELSE
        CURRENT_CELL_X <= CURRENT_CELL_X + 1;
        GRID_X <= CURRENT_CELL_X - 2; -- INITIATE ADDRESS POINTING AT THE FIRST CELL OF THE NEXT 7X7 NEIGHBORHOOD.
        GRID_Y <= CURRENT_CELL_Y - 1920;
     END IF;
    WHITE_COUNTER <= (OTHERS => '0'); -- INITIATE COUNTERS BUT NOT FLAGS!
    GREY_COUNTER <= (OTHERS => '0');
    RED_COUNTER <= (OTHERS => '0');
    GREEN_COUNTER <= (OTHERS => '0');
    CENTER_MOORE_NEIGHBORHOOD <= '0';
    PARK_MOORE_NEIGHBORHOOD <= '0';
    MOORE_NEIGHBORHOOD_EMPTY <= '1';
    Y <= READ;
WHEN WAIT_EN => -- DON'T GO TO THE NEXT FRAME UNTIL EN = 1. WE WANT THIS IN ORDER TO CONTROL THE CA'S SPEED.
    IF (EN = '1') THEN
         Y <= EX_MEM;
```

```
                        ELSE
                            Y <= Y;
                         END IF;
                    WHEN EX_MEM => -- EXCHANGE BRAMS -- INITIATE COUNTERS BUT NOT FLAGS!
                        SEL_SIG <= NOT SEL_SIG;
                        CURRENT_CELL_X <= "0000000011"; -- CELL 3  (4TH CELL)          3 X 3 THE FIRST CELL ON THE GRID WITH A 7X7 EXTENDED MOORE
                                                                                       NEIGHBORHOOD
                        CURRENT_CELL_Y <= "000000011110000000"; -- IN ROW 3 (3*640=1920) (4TH ROW)
                        GRID_X <= (OTHERS => '0'); -- INITIATE ADDRESS 0, FIRST CELL OF THE GRID - FIRST CELL OF THE FIRST 7X7 NEIGHBORHOOD
                        GRID_Y <= (OTHERS => '0');
                        WHITE_COUNTER <= (OTHERS => '0');
                        GREY_COUNTER <= (OTHERS => '0');
                        RED_COUNTER <= (OTHERS => '0');
                        GREEN_COUNTER <= (OTHERS => '0');
                        CENTER_MOORE_NEIGHBORHOOD <= '0';
                        PARK_MOORE_NEIGHBORHOOD <= '0';
                        MOORE_NEIGHBORHOOD_EMPTY <= '1';
                        Y <= READ;
                END CASE;
            END IF;
        END PROCESS CA_RULES;

    ADDRESS <= GRID_X + GRID_Y;
    SEL <= SEL_SIG;

END BEHAVIORAL;
```

```vhdl
----------------------------------------------------------------------------------
   -- TECHNICAL UNIVERSITY OF CRETE
   -- NICK KYPARISSAS
   --
   -- CREATE DATE: 16 JUNE 2015
   -- MODULE NAME: GRAPHICS_CTRL - BEHAVIORAL
   -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
   --
   -- DESCRIPTION: GRAPHICS CONTROLLER'S TOP LEVEL MODULE. IT CONTAINS
   -- THE WXGA CONTROLLER AND THE COLORS CONTROLLER
----------------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY GRAPHICS_CTRL IS
   PORT (
       CLK      : IN STD_LOGIC;  -- @ 83.46 MHZ
       RST      : IN STD_LOGIC;
       MEM_DATA     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
       HS          : OUT STD_LOGIC;
       VS          : OUT STD_LOGIC;
       R           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       G           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       B           : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       MEM_ADDRESS : OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
   );
END GRAPHICS_CTRL;

ARCHITECTURE BEHAVIORAL OF GRAPHICS_CTRL IS
    --SIGNALS
    SIGNAL HCWX_SIGNAL          : STD_LOGIC_VECTOR(10 DOWNTO 0) := (OTHERS => '0');
    SIGNAL VCWX_SIGNAL          : STD_LOGIC_VECTOR(10 DOWNTO 0) := (OTHERS => '0');

    --COMPONENTS
    COMPONENT WXGA_CONTROLLER
        PORT(
            RST         : IN STD_LOGIC;
            CLK         : IN STD_LOGIC; -- MUST BE @ 83.46 MHZ
            HS          : OUT STD_LOGIC;
            VS          : OUT STD_LOGIC;
            HCOUNT      : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);
            VCOUNT      : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
        );
   END COMPONENT;

    COMPONENT WXGA_COLOR_CTRL
        PORT (
            RST         : IN STD_LOGIC;
            CLK         : IN STD_LOGIC;
            HCOUNT      : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
            VCOUNT      : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
            MEM_DATA    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            MEM_ADDRESS : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
            RED         : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            GREEN       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            BLUE        : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
   END COMPONENT;

   BEGIN

   WXGA_COLOR_CONTROLLER : WXGA_COLOR_CTRL
        PORT MAP(
            RST => RST,
            CLK => CLK,
            HCOUNT => HCWX_SIGNAL,
            VCOUNT => VCWX_SIGNAL,
            MEM_DATA => MEM_DATA,
            MEM_ADDRESS => MEM_ADDRESS,
            RED => R,
            GREEN => G,
            BLUE => B
        );

   WXGA_SYNC_CONTROLLER : WXGA_CONTROLLER
        PORT MAP(
            RST => RST,
            CLK => CLK,
            HS => HS,
            VS => VS,
            HCOUNT => HCWX_SIGNAL,
            VCOUNT => VCWX_SIGNAL
        );

END BEHAVIORAL;
```

```vhdl
----------------------------------------------------------------------------------
   -- TECHNICAL UNIVERSITY OF CRETE
   -- NICK KYPARISSAS
   --
   -- CREATE DATE: 16 JUNE 2015
   -- MODULE NAME: WXGA_COLOR_CTRL - BEHAVIORAL
   -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
   --
   -- DESCRIPTION: THIS MODULE CONTROLS WHAT COLORS THE WXGA CONTROLLER WILL DISPLAY ON THE SCREEN.
   -- TECHNICALLY, IT SCALES UP A 640X400 IMAGE FROM BRAM, TO A 1280X800 FRAME.
----------------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY WXGA_COLOR_CTRL IS
    PORT (
        RST         : IN STD_LOGIC;
        CLK         : IN STD_LOGIC;
        HCOUNT      : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        VCOUNT      : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        MEM_DATA    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        MEM_ADDRESS : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
        RED         : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        GREEN       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        BLUE        : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
   );
END WXGA_COLOR_CTRL;

ARCHITECTURE BEHAVIORAL OF WXGA_COLOR_CTRL IS

   SIGNAL COUNTER2 : STD_LOGIC; -- 2 PIXELS OF A SCREEN ROW EQUAL TO 1 PIXEL OF THE 640X400 IMAGE
   SIGNAL COUNTER640 : STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0'); -- IN WHICH PIXEL OF THE IMAGE ROW WE ARE
   SIGNAL COUNTER2LINES : STD_LOGIC; -- EACH PIXEL OF THE IMAGE EQUALS TO A 2X2 PIXEL WXGA SCREEN
   SIGNAL COUNTERIMAGELINE : STD_LOGIC_VECTOR(17 DOWNTO 0) := (OTHERS => '0'); -- IN WHICH PIXEL ROW OF THE IMAGE WE ARE

BEGIN

    PROCESS
        BEGIN
        WAIT UNTIL CLK'EVENT AND CLK= '1';
        IF (RST = '1') THEN -- ABOUT TO START A NEW FRAME, INITIALIZE ALL COUNTERS.
            COUNTER2 <= '0';
            COUNTER640 <= (OTHERS => '0');
            COUNTER2LINES <= '0';
            COUNTERIMAGELINE <= (OTHERS => '0');
            RED <= "0000";
            GREEN <= "0000";
            BLUE <= "0000";
        ELSE
            IF (HCOUNT < 1280 AND VCOUNT < 800) THEN --
                COUNTER2 <= NOT COUNTER2;
                IF (COUNTER2 = '1') THEN
                    IF (COUNTER640 = 639) THEN
                        COUNTER640 <= (OTHERS => '0');
                        COUNTER2LINES <= NOT COUNTER2LINES;
                        IF (COUNTER2LINES = '1') THEN
                            COUNTERIMAGELINE <= COUNTERIMAGELINE + 640; -- CHANGE IMAGE LINE.
                        END IF;
                        ELSE
                            COUNTER640 <= COUNTER640 + 1;
                        END IF;
                END IF;
                RED <= MEM_DATA(7 DOWNTO 5)&'0';
                GREEN <= MEM_DATA(4 DOWNTO 2)&'0';
                BLUE <= MEM_DATA(1 DOWNTO 0)&'0'&'0';
            ELSIF (HCOUNT = 1680 AND VCOUNT = 828) THEN -- ABOUT TO START A NEW FRAME, INITIALIZE ALL COUNTERS.
                COUNTER2 <= '0';
                COUNTER640 <= (OTHERS => '0');
                COUNTER2LINES <= '0';
                COUNTERIMAGELINE <= (OTHERS => '0');
                RED <= "0000";
                GREEN <= "0000";
                BLUE <= "0000";
            ELSE
                COUNTER2 <= '0';
                COUNTER640 <= (OTHERS => '0');
                RED <= "0000";
                GREEN <= "0000";
                BLUE <= "0000";
            END IF;
        END IF;
   END PROCESS;

    MEM_ADDRESS <= COUNTERIMAGELINE + COUNTER640;

END BEHAVIORAL;
```

```vhdl
----------------------------------------------------------------------------------
    -- TECHNICAL UNIVERSITY OF CRETE
    -- NICK KYPARISSAS
    --
    -- CREATE DATE: 16 JUNE 2015
    -- MODULE NAME: WXGA_CONTROLLER - BEHAVIORAL
    -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
    --
    -- DESCRIPTION: THIS MODULE GENERATES THE VIDEO SYNCH PULSES FOR THE MONITOR TO
    -- ENTER 1280X800@60HZ RESOLUTION STATE. IT ALSO PROVIDES HORIZONTAL
    -- AND VERTICAL COUNTERS FOR THE CURRENTLY DISPLAYED PIXEL AND A BLANK
    -- SIGNAL THAT IS ACTIVE WHEN THE PIXEL IS NOT INSIDE THE VISIBLE SCREEN
    -- AND THE COLOR OUTPUTS SHOULD BE RESET TO 0.
    --
    -- BASED ON ULRICH ZOLTAN'S VGA CONTROLLER, COPYRIGHT 2006 DIGILENT, INC.
----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

-- SIMULATION LIBRARY
LIBRARY UNISIM;
USE UNISIM.VCOMPONENTS.ALL;

ENTITY WXGA_CONTROLLER IS
    PORT(
        RST         : IN STD_LOGIC;
        CLK         : IN STD_LOGIC; -- MUST BE @ 83.46 MHZ
        HS          : OUT STD_LOGIC;
        VS          : OUT STD_LOGIC;
        HCOUNT      : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);
        VCOUNT      : OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
    );
END WXGA_CONTROLLER;

ARCHITECTURE BEHAVIORAL OF WXGA_CONTROLLER IS

    ------------------------------------------------------------------------
        -- CONSTANTS
    ------------------------------------------------------------------------

    -- MAXIMUM VALUE FOR THE HORIZONTAL PIXEL COUNTER
    CONSTANT HMAX  : STD_LOGIC_VECTOR(10 DOWNTO 0) := "11010010000"; -- 1680 TOTAL PIXELS PER LINE
    -- MAXIMUM VALUE FOR THE VERTICAL PIXEL COUNTER
    CONSTANT VMAX  : STD_LOGIC_VECTOR(10 DOWNTO 0) := "01100111100"; -- 828 TOTAL LINES
    -- TOTAL NUMBER OF VISIBLE COLUMNS
    CONSTANT HLINES: STD_LOGIC_VECTOR(10 DOWNTO 0) := "10100000000"; -- 1280 RESOLUTION WIDTH
    -- VALUE FOR THE HORIZONTAL COUNTER WHERE FRONT PORCH ENDS
    CONSTANT HFP   : STD_LOGIC_VECTOR(10 DOWNTO 0) := "10101000000"; -- 1344 = FRONT PORCH 64 PIXELS + 1280
    -- VALUE FOR THE HORIZONTAL COUNTER WHERE THE SYNCH PULSE ENDS
    CONSTANT HSP   : STD_LOGIC_VECTOR(10 DOWNTO 0) := "10111001000"; -- 1480 = HORIZONTAL SYNC 136 PIXELS + 1344
    -- TOTAL NUMBER OF VISIBLE LINES
    CONSTANT VLINES: STD_LOGIC_VECTOR(10 DOWNTO 0) := "01100100000"; -- 800 RESOLUTION HEIGHT
    -- VALUE FOR THE VERTICAL COUNTER WHERE THE FRONT PORCH ENDS
    CONSTANT VFP   : STD_LOGIC_VECTOR(10 DOWNTO 0) := "01100100001"; -- 801 = FRONT PORCH 1 LINES + 800
    -- VALUE FOR THE VERTICAL COUNTER WHERE THE SYNCH PULSE ENDS
    CONSTANT VSP   : STD_LOGIC_VECTOR(10 DOWNTO 0) := "01100111000"; -- 824 = VERTICAL SYNC 3 LINES + 821
    -- POLARITY OF THE HORIZONTAL AND VERTICAL SYNCH PULSE
    CONSTANT HSP_P   : STD_LOGIC := '0';
    CONSTANT VSP_P   : STD_LOGIC := '1';
    ------------------------------------------------------------------------
        -- SIGNALS
    ------------------------------------------------------------------------

    -- HORIZONTAL AND VERTICAL COUNTERS
    SIGNAL HCOUNTER : STD_LOGIC_VECTOR(10 DOWNTO 0) := (OTHERS => '0');
    SIGNAL VCOUNTER : STD_LOGIC_VECTOR(10 DOWNTO 0) := (OTHERS => '0');

    ------------------------------------------------------------------------

BEGIN

    -- OUTPUT HORIZONTAL AND VERTICAL COUNTERS.
    HCOUNT <= HCOUNTER;
    VCOUNT <= VCOUNTER;

    -- INCREMENT HORIZONTAL COUNTER AT CLK RATE
    -- UNTIL HMAX IS REACHED, THEN RESET AND KEEP COUNTING.
    H_COUNT: PROCESS
        BEGIN
        WAIT UNTIL CLK'EVENT AND CLK = '1' ;

        IF(RST = '1') THEN
            HCOUNTER <= (OTHERS => '0');
        ELSIF(HCOUNTER = HMAX) THEN
            HCOUNTER <= (OTHERS => '0');
        ELSE
            HCOUNTER <= HCOUNTER + 1;
        END IF;
    END PROCESS H_COUNT;
    -- INCREMENT VERTICAL COUNTER WHEN ONE LINE IS FINISHED
    -- (HORIZONTAL COUNTER REACHED HMAX)
    -- UNTIL VMAX IS REACHED, THEN RESET AND KEEP COUNTING.
    V_COUNT: PROCESS
        BEGIN
        WAIT UNTIL CLK'EVENT AND CLK= '1';

        IF(RST = '1') THEN
            VCOUNTER <= (OTHERS => '0');
        ELSIF(HCOUNTER = HMAX) THEN
            IF(VCOUNTER = VMAX) THEN
                VCOUNTER <= (OTHERS => '0');
            ELSE
                VCOUNTER <= VCOUNTER + 1;
            END IF;
        END IF;
    END PROCESS V_COUNT;

    -- GENERATE HORIZONTAL SYNCH PULSE
    -- WHEN HORIZONTAL COUNTER IS BETWEEN WHERE THE
    -- FRONT PORCH ENDS AND THE SYNCH PULSE ENDS.
    -- THE HS IS ACTIVE (WITH POLARITY HSP_P) FOR A TOTAL OF 136 PIXELS.
    DO_HS: PROCESS
        BEGIN
        WAIT UNTIL CLK'EVENT AND CLK='1';

        IF(HCOUNTER >= HFP AND HCOUNTER < HSP) THEN
            HS <= HSP_P;
        ELSE
            HS <= NOT HSP_P;
        END IF;
    END PROCESS DO_HS;

    -- GENERATE VERTICAL SYNCH PULSE
    -- WHEN VERTICAL COUNTER IS BETWEEN WHERE THE
    -- FRONT PORCH ENDS AND THE SYNCH PULSE ENDS.
    -- THE VS IS ACTIVE (WITH POLARITY VSP_P) FOR A TOTAL OF 3 LINES.
    DO_VS: PROCESS
        BEGIN
        WAIT UNTIL CLK'EVENT AND CLK='1';

        IF(VCOUNTER >= VFP AND VCOUNTER < VSP) THEN
            VS <= VSP_P;
        ELSE
            VS <= NOT VSP_P;
        END IF;
    END PROCESS DO_VS;

END BEHAVIORAL;
```

```vhdl
----------------------------------------------------------------------------------
   -- TECHNICAL UNIVERSITY OF CRETE
   -- NICK KYPARISSAS
   --
   -- CREATE DATE: 16 JUNE 2015
   -- MODULE NAME: SPEED_CONTROLLER - BEHAVIORAL
   -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
   --
   -- DESCRIPTION: THIS MODULE CONTROLS WHETHER THE CA WILL BE WORKING AUTOMATICALLY @ FULL SPEED
   -- OR MANUALLY, WAITING FOR THE USER TO DECIDE WHEN IT WILL MOVE ON.
----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SPEED_CONTROLLER IS
    PORT (  UP                  : IN STD_LOGIC;
            PAUSE_STEP          : IN STD_LOGIC;
            CLK             : IN STD_LOGIC;
            RST             : IN STD_LOGIC;
            EN                  : OUT STD_LOGIC;
            SPEED_INDICATOR  : OUT STD_LOGIC);
END SPEED_CONTROLLER;

ARCHITECTURE BEHAVIORAL OF SPEED_CONTROLLER IS
    TYPE STATE IS (AUTO, MANUAL_STEP);
    SIGNAL Y : STATE;

BEGIN
   PROCESS
        BEGIN
        WAIT UNTIL CLK'EVENT AND CLK = '1';

        IF ( RST = '1' ) THEN
            SPEED_INDICATOR <= '0';
            EN <= '0';
            Y <= MANUAL_STEP;
        ELSE
            CASE Y IS
                WHEN MANUAL_STEP =>
                    IF (UP = '1') THEN
                        Y <= AUTO;
                    END IF;
                    SPEED_INDICATOR <= '0';
                    EN <= PAUSE_STEP;
                WHEN AUTO =>
                    EN <= '1';
                    IF (PAUSE_STEP = '1') THEN
                        Y <= MANUAL_STEP;
                    END IF;
                    SPEED_INDICATOR <= '1';
            END CASE;
        END IF;
   END PROCESS;
END BEHAVIORAL;
```

```vhdl
--------------------------------------------------------------------------------
    -- TECHNICAL UNIVERSITY OF CRETE
    -- NICK KYPARISSAS
    --
    -- CREATE DATE: 16 JUNE 2015
    -- MODULE NAME: SEVSEG_CONTROLLER - BEHAVIORAL
    -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
    --
    -- DESCRIPTION: A MODULE THAT MULTIPLEXES 4 DIFFERENT 7-SEGMENT DIGITS.
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY SEVSEG_CONTROLLER IS
    PORT ( CLOCK          : IN  STD_LOGIC;
        RESET             : IN  STD_LOGIC;
        INPUT             : IN  STD_LOGIC;
        SEVEN_SEG_EN      : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- WHICH 7-SEGMENT DISPLAY DIGIT WILL BE ENABLED,
        SEVEN_SEG_DATA    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); -- AND WHAT IT WILL BE DISPLAYING.
END SEVSEG_CONTROLLER;

ARCHITECTURE BEHAVIORAL OF SEVSEG_CONTROLLER IS

    TYPE STATE IS (PRINTNUM1, PRINTNUM2, PRINTNUM3, PRINTNUM4); -- ACT ON ONE DIGIT AT A TIME.
    SIGNAL Y : STATE;
    SIGNAL SEVEN_SEG_EN_SIG : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL COUNTER : INTEGER RANGE 0 TO 138000 := 0;

BEGIN
    PROCESS
        BEGIN
        WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';

        IF ( RESET = '1' ) THEN
            COUNTER <= 0;
            Y <= PRINTNUM1 ;
        ELSE
            CASE Y IS
                WHEN PRINTNUM1 =>
                    SEVEN_SEG_EN_SIG <= "11101111";
                    IF (INPUT = '0') THEN
                        SEVEN_SEG_DATA <= "00110001"; -- P
                    ELSE
                        SEVEN_SEG_DATA <= "00000011"; -- O
                    END IF;
                    IF ( COUNTER = 138000 ) THEN -- 138000 / 83 MHZ, ENOUGH FOR THE HUMAN EYE TO SEE NO FLICKERING.
                        COUNTER <= 0;
                        Y <= PRINTNUM2;
                    ELSE
                        Y <= Y;
                        COUNTER <= COUNTER + 1;
                    END IF;
                WHEN PRINTNUM2 =>
                    SEVEN_SEG_EN_SIG <= "11011111";
                    IF (INPUT = '0') THEN
                        SEVEN_SEG_DATA <= "01100001"; -- E
                    ELSE
                        SEVEN_SEG_DATA <= "11100001"; -- T
                    END IF;
                    IF ( COUNTER = 138000 ) THEN
                        COUNTER <= 0 ;
                        Y <= PRINTNUM3;
                    ELSE
                        Y <= Y;
                        COUNTER <= COUNTER + 1 ;
                    END IF;
                WHEN PRINTNUM3 =>
                    SEVEN_SEG_EN_SIG <= "10111111";
                    IF (INPUT = '0') THEN
                        SEVEN_SEG_DATA <= "11100001"; -- T
                    ELSE
                        SEVEN_SEG_DATA <= "10000011"; -- U
                    END IF;
                    IF ( COUNTER = 138000 ) THEN
                        COUNTER <= 0 ;
                        Y <= PRINTNUM4;
                    ELSE
                        Y <= Y;
                        COUNTER <= COUNTER + 1 ;
                    END IF;
                WHEN PRINTNUM4 =>
                    SEVEN_SEG_EN_SIG <= "01111111";
                    IF (INPUT = '0') THEN
                        SEVEN_SEG_DATA <= "01001001"; -- S
                    ELSE
                        SEVEN_SEG_DATA <= "00010001"; -- A
                    END IF;
                    IF ( COUNTER = 138000 ) THEN
                        COUNTER <= 0 ;
                        Y <= PRINTNUM1;
                    ELSE
                        Y <= Y;
                        COUNTER <= COUNTER + 1 ;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;
    SEVEN_SEG_EN <= SEVEN_SEG_EN_SIG WHEN RESET = '0' ELSE "11111111"; -- IN ORDER TO AVOID "ARTIFACT" DIGITS SHOWN ON THE 7-SEG DISPLAY FROM
                                                                        -- PREVIOUS STAGES WHEN "RESET" REMAINS 1 FOR LONG ENOUGH FOR THE HUMAN
                                                                        -- EYE TO CATCH IT.

END BEHAVIORAL;
```

```
----------------------------------------------------------------------------------
   -- TECHNICAL UNIVERSITY OF CRETE
   -- NICK KYPARISSAS
   --
   -- CREATE DATE: 16 JUNE 2015
   -- MODULE NAME: DEBOUNCE - BEHAVIORAL
   -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
   --
   -- DESCRIPTION: A 2-LEVEL DEBOUNCER. BASED ON www.eewiki.net 'S EXAMPLE DESIGNS.
----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY DEBOUNCE IS
   PORT(
       CLOCK        : IN  STD_LOGIC;  -- INPUT CLOCK.
       BUTTON   : IN  STD_LOGIC;  -- INPUT SIGNAL TO BE DEBOUNCED.
       RESULT   : OUT STD_LOGIC); -- DEBOUNCED SIGNAL.
END DEBOUNCE;

ARCHITECTURE BEHAVIORAL OF DEBOUNCE IS
   SIGNAL FLIPFLOP    : STD_LOGIC_VECTOR(1 DOWNTO 0); -- INPUT FLIP FLOPS.
   SIGNAL COUNTER_SET : STD_LOGIC;                    -- SYNC RESET TO ZERO.
   SIGNAL COUNTER_OUT : STD_LOGIC_VECTOR(20 DOWNTO 0) := (OTHERS => '0'); -- COUNTER OUTPUT // COUNTER SIZE (20 BITS GIVES 20 MSEC WITH 50MHZ
                                                                           CLOCK).

BEGIN

   COUNTER_SET <= FLIPFLOP(0) XOR FLIPFLOP(1);   -- DETERMINE WHEN TO START/RESET COUNTER.

   PROCESS
       BEGIN
       WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';

       FLIPFLOP(0) <= BUTTON;
       FLIPFLOP(1) <= FLIPFLOP(0); -- PASSING ALONG THE SIGNAL.
       IF(COUNTER_SET = '1') THEN --RESET COUNTER WHEN INPUT IS CHANGING.
           COUNTER_OUT <= (OTHERS => '0');
       ELSIF(COUNTER_OUT(20) = '0') THEN -- STABLE INPUT TIME IS NOT YET MET.
           COUNTER_OUT <= COUNTER_OUT + 1;
       ELSE --STABLE INPUT TIME IS MET.
           RESULT <= FLIPFLOP(1);
       END IF;
   END PROCESS;

END BEHAVIORAL;
```

```
----------------------------------------------------------------------------------
   -- TECHNICAL UNIVERSITY OF CRETE
   -- NICK KYPARISSAS
   --
   -- CREATE DATE: 16 JUNE 2015
   -- MODULE NAME: MUX_2_IN_1_8B - BEHAVIORAL
   -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
   --
   -- DESCRIPTION: A TYPICAL MUXER.
----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX_2_IN_1_8B IS
    PORT ( INPUT00  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
           INPUT01  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
           SEL      : IN STD_LOGIC;
           OUTPUT   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END MUX_2_IN_1_8B;

ARCHITECTURE BEHAVIORAL OF MUX_2_IN_1_8B IS

BEGIN

OUTPUT <= INPUT00 WHEN SEL = '0' ELSE
            INPUT01 WHEN SEL = '1' ;

END BEHAVIORAL;
```

```vhdl
----------------------------------------------------------------------------------
    -- TECHNICAL UNIVERSITY OF CRETE
    -- NICK KYPARISSAS
    --
    -- CREATE DATE: 16 JUNE 2015
    -- MODULE NAME: DEMUX_1_IN_2_1B - BEHAVIORAL
    -- PROJECT NAME: GAME OF COMPLEX LIFE - XILINX OPEN HARDWARE 2015 PARTICIPATION
    --
    -- DESCRIPTION: A TYPICAL DEMUXER.
----------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DEMUX_1_IN_2_1B IS
    PORT ( INPUT    : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
           SEL      : IN STD_LOGIC;
           OUTPUT0  : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
           OUTPUT1  : OUT STD_LOGIC_VECTOR (0 DOWNTO 0));
END DEMUX_1_IN_2_1B;

ARCHITECTURE BEHAVIORAL OF DEMUX_1_IN_2_1B IS

BEGIN

OUTPUT0 <= INPUT WHEN SEL = '1' ELSE "0"; -- SEL (DEMUXER'S INPUT @ TOP LEVEL) INVERTED HERE, UNLIKE THE MUXERS OF THE DESIGNS.
OUTPUT1 <= INPUT WHEN SEL = '0' ELSE "0";

END BEHAVIORAL;
```

```vhdl
----------------------------------------------------------------------------------------------------------------------
    -- COMPANY: TECHNICAL UNIVERSITY OF CRETE
    -- ENGINEER: NICK KYPARISSAS
    --
    -- CREATE DATE: 14 JUNE 2015
    -- DESIGN NAME:
    -- MODULE NAME: CELLULAR_AUTOMATON_MODULE - BEHAVIORAL
    -- TARGET DEVISES: SYNTHESIZABLE, A VARIATION HAS BEEN TESTED ON NEXYS 4.
    --
    -- DESCRIPTION: THIS MODULE LOADS AN EXANDING CELLULAR AUTOMATON (CA) FROM MEMORY MODULE NO. 1,
    -- ACTS ON IT ACCORDING TO THE CA'S RULES AND STORES THE RESULT IN MEMORY MODULE NO. 2.
    -- THEN IT EXCHANGES THE MEMORY MODULES, READS FROM MEMORY MODULE NO. 2 AND WRITES IN MEMORY MODULE NO. 1.
    -- DUE TO LIMITED MEMORY SIZE, I HAD TO AVOID HAVING DYING CELLS (BECAUSE THEN YOU NEED MORE MEMORY TO
    -- STORE A 3RD FRAME WITHOUT ANY CELLS ON IT), SO THIS MODULE IMPLEMENTS A FOREVER EXPANDING CA.
    --
    -- IMPLEMENTED FOR XILINX UNIVERSITY PROGRAM'S "OPEN HARDWARE 2015" COMPETITION.
----------------------------------------------------------------------------------------------------------------------


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY CELLULAR_AUTOMATON_MODULE IS
    GENERIC(
        GRID_SIZE_X         :  INTEGER := 100; -- USUALLY FITS INTO A SCREEN RESOLUTION, A MEMORY OR A SUBDIVISION OF THOSE.
        GRID_SIZE_Y         :  INTEGER := 100;
        NEIGHBORHOOD_SIZE   :  INTEGER := 3 -- MOORE NEIGHBORHOOD SIZE: USE "3" FOR THE CLASSIC 3X3 MOORE NEIGHBORHOOD.
    );
    PORT ( DATA_IN          : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- DATA SIZE, HERE: A BYTE.
        EN                  : IN STD_LOGIC; -- WHEN EN COMES, IT EXCHANGES BETWEEN THE TWO MEMORY MODULES.
        RST                 : IN STD_LOGIC;
        CLOCK               : IN STD_LOGIC;
        WEN                 : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        SEL                 : OUT STD_LOGIC;
        ADDRESS             : OUT STD_LOGIC_VECTOR (13 DOWNTO 0); -- IT NEEDS TO FIT THE GRID AND THE MEMORY.
        DATA_OUT            : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END CELLULAR_AUTOMATON_MODULE;

ARCHITECTURE BEHAVIORAL OF CELLULAR_AUTOMATON_MODULE IS

    TYPE STATE IS (RESET, READ, STORE, CALCULATE_CA_INITIALIZATION, CALCULATE_CA, WRITE, CALCULATE_NEXT_CELL, WAIT_EN, EX_MEM);
    SIGNAL Y                : STATE;
    SIGNAL CURRENT_CELL_X   : STD_LOGIC_VECTOR(6 DOWNTO 0); -- IT NEEDS TO FIT THE GRID_SIZE_X'S MAXIMUM VALUE. SET IT AS THE CENTRAL CELL'S
                                                            -- X DIMENSION OF THE FIRST FULL NEIGHBORHOOD OF YOUR GRID. FOR EXAMPLE, FOR A
                                                            -- 7X7 GRID, IT'S VALUE WOULD BE 3.
    SIGNAL CURRENT_CELL_Y   : STD_LOGIC_VECTOR(13 DOWNTO 0); -- IT NEEDS TO FIT THE (GRID_SIZE_X * GRID_SIZE_Y)'S MAXIMUM VALUE.
    SIGNAL GRID_X           : STD_LOGIC_VECTOR(6 DOWNTO 0); -- DIMENSIONS OF THE CELL TO BE READ OR WRITTEN. AT FIRST, IT HAS TO BE THE FIRST
                                                            -- CELL OF THE NEIGHBORHOOD.
    SIGNAL GRID_Y           : STD_LOGIC_VECTOR(13 DOWNTO 0);
    SIGNAL COUNTER          : STD_LOGIC_VECTOR(3 DOWNTO 0); -- IT NEEDS TO FIT NEIGHBORHOOD_SIZE^2.
    SIGNAL SEL_SIG          : STD_LOGIC;

BEGIN

    CA_RULES: PROCESS
        BEGIN

        WAIT UNTIL CLOCK'EVENT AND CLOCK = '1';

        IF ( RST = '1' ) THEN
        Y <= RESET;
        ELSE
            CASE Y IS
                WHEN RESET =>
                    -- OUTPUTS:
                    SEL_SIG <= '0'; -- SELECT THE FIRST RAM MODULE.
                    WEN <= "0"; -- DO NOT WRITE ANYTHING.
                    -- THE OTHER OUTPUTS' VALUES ARE IRRELEVANT HERE.
                    -- SIGNALS:
                    CURRENT_CELL_X <= "0000001"; -- THE CENTRAL CELL'S X DIMENSION OF THE FIRST FULL NEIGHBORHOOD, HERE: 1X1.
                    CURRENT_CELL_Y <= "00000001100100"; -- CURRENT_CELL_Y IS COMPUTED AS: CURRENT ROW'S NUMBER * GRID_SIZE_X.
                    GRID_X <= (OTHERS => '0'); -- INITIATE ADDRESS 0, FIRST CELL OF THE GRID - FIRST CELL OF THE FIRST NEIGHBORHOOD.
                    GRID_Y <= (OTHERS => '0');
                    COUNTER <= (OTHERS => '0');
                    -- NEXT STATE:
                    Y <= READ;
                WHEN STORE =>
                    IF (GRID_X = CURRENT_CELL_X + NEIGHBORHOOD_SIZE/2) AND (GRID_Y = CURRENT_CELL_Y + GRID_SIZE_X*NEIGHBORHOOD_SIZE/2) THEN
                    --IF THE FINAL CELL OF THE NEIGHBORHOOD.
                        GRID_X <= CURRENT_CELL_X;
                        GRID_Y <= CURRENT_CELL_Y;
                        Y <= CALCULATE_CA_INITIALIZATION;
                    ELSE -- CALCULATE NEXT ADDRESS.
                        IF (GRID_X = CURRENT_CELL_X + NEIGHBORHOOD_SIZE/2) THEN
                            GRID_X <= CURRENT_CELL_X - NEIGHBORHOOD_SIZE/2; -- THE FIRST CELL OF THE NEXT ROW OF THE NEIGHBORHOOD.
                            GRID_Y <= GRID_Y + GRID_SIZE_X;
                        ELSE
                            GRID_X <= GRID_X + 1;
                        END IF;
                        Y <= READ;
                    END IF;
                    -- CHECK THE MOORE NEIGHBORHOOD (NOT THE EXTENDED ONE, BUT THE NORMAL 3X3 NEIGHBORHOOD).
                    IF (GRID_Y > CURRENT_CELL_Y - 2*GRID_SIZE_X) AND (GRID_Y < CURRENT_CELL_Y + 2*GRID_SIZE_X)
                    AND (GRID_X > CURRENT_CELL_X - 2) AND (GRID_X < CURRENT_CELL_X + 2) THEN
                        IF (DATA_IN /= "00000000") THEN --IF WE HAVE DATA IN THE NEIGHBORHOOD.
                            COUNTER <= COUNTER + 1;
                        END IF;
                    END IF;
                    -- STORE THE APPROPRIATE COUNTER VALUES.
                    IF (GRID_X /= CURRENT_CELL_X) AND (GRID_Y /= CURRENT_CELL_Y) THEN -- IF WE ARE NOT READING THE CURRENT CELL.
                        IF (DATA_IN /= "00000000") THEN -- IF WE HAVE DATA IN THE EXTENDED NEIGHBORHOOD. YOU CAN INCREASE DIFFERENT
                                                        -- COUNTERS FOR DIFFERENT TYPES OF DATA!
                            COUNTER <= COUNTER + 1;
                        END IF;
                    END IF;
                WHEN READ =>   -- THE ADDRESSES HERE HAVE BEEN SET. IN THE NEXT CYCLE, THE DATA WILL HAVE BEEN SET ON THE DATA BUS.
                    Y <= STORE;
                WHEN CALCULATE_CA_INITIALIZATION => -- THE ADDRESSES HERE HAVE BEEN SET. IN THE NEXT CYCLE, THE DATA WILL HAVE BEEN
                                                    -- SET ON THE DATA BUS.
                    Y <= CALCULATE_CA;
                WHEN CALCULATE_CA =>
                    IF (COUNTER > 3) THEN -- IF THE RULES APPLY, FOR EXAMPLE HERE: COUNTER > 3
                        DATA_OUT <= "10101010"; -- WRITE WHAT THE RULES SAY.
                    ELSE
                        DATA_OUT <= DATA_IN;
                    END IF;
                    WEN <= "1";
                    Y <= WRITE;
                WHEN WRITE => -- IF WE ARE IN THE END OF THE FRAME.
                    IF ((CURRENT_CELL_X = GRID_SIZE_X - NEIGHBORHOOD_SIZE/2) AND (CURRENT_CELL_Y = GRID_SIZE_X*(GRID_SIZE_Y-1))) THEN
                       -- FINAL CELL
                        Y <= WAIT_EN; -- THIS "WAIT" STATE HAS BEEN SET IN CASE YOU WANT TO CONTROL THE CA'S SPEED.
                                      -- IF YOU DON'T WANT THAT, JUMP TO "EX_MEM".
                    ELSE
                        Y <= CALCULATE_NEXT_CELL;
                    END IF;
                    WEN <= "0";
                WHEN CALCULATE_NEXT_CELL =>
                    IF (CURRENT_CELL_X = GRID_SIZE_X - NEIGHBORHOOD_SIZE/2) THEN
                        CURRENT_CELL_X <= "0000001"; -- THE CENTRAL CELL'S X DIMENSION OF THE FIRST FULL NEIGHBORHOOD, HERE: 1X1.
                        CURRENT_CELL_Y <= CURRENT_CELL_Y + GRID_SIZE_X; --CHANGE LINE
                        GRID_X <= (OTHERS => '0'); -- INITIATE ADDRESS POINTING AT THE FIRST CELL OF THE NEXT NEIGHBORHOOD.
                        GRID_Y <= STD_LOGIC_VECTOR(TO_UNSIGNED(NEIGHBORHOOD_SIZE/2, GRID_Y'LENGTH));
                    ELSE
                        CURRENT_CELL_X <= CURRENT_CELL_X + 1;
                        GRID_X <= CURRENT_CELL_X - NEIGHBORHOOD_SIZE/2 -1; -- INITIATE ADDRESS POINTING AT THE FIRST CELL OF
                                                                           -- THE NEXT NEIGHBORHOOD.
                        GRID_Y <= CURRENT_CELL_Y - GRID_SIZE_X*NEIGHBORHOOD_SIZE/2;
                    END IF;
                    COUNTER <= (OTHERS => '0');
                    Y <= READ;
                WHEN WAIT_EN => -- THIS "WAIT" STATE HAS BEEN SET IN CASE YOU WANT TO CONTROL THE CA'S SPEED.
                    IF (EN = '1') THEN
                        Y <= EX_MEM;
                    ELSE
                        Y <= Y;
                    END IF;
                WHEN EX_MEM =>
                    SEL_SIG <= NOT SEL_SIG; -- NEXT FRAME, EXCHANGE MEMORIES.
                    -- SIGNALS:
                    CURRENT_CELL_X <= "0000001"; -- THE CENTRAL CELL'S X DIMENSION OF THE FIRST FULL NEIGHBORHOOD, HERE: 1X1.
                    CURRENT_CELL_Y <= "00000001100100"; -- CURRENT_CELL_Y IS COMPUTED AS: CURRENT ROW'S NUMBER * GRID_SIZE_X.
                    GRID_X <= (OTHERS => '0'); -- INITIATE ADDRESS 0, FIRST CELL OF THE GRID - FIRST CELL OF THE FIRST NEIGHBORHOOD
                    GRID_Y <= (OTHERS => '0');
                    COUNTER <= (OTHERS => '0');
                    Y <= READ;
            END CASE;
        END IF;
    END PROCESS CA_RULES;

    ADDRESS <= GRID_X + GRID_Y;
    SEL <= SEL_SIG;

END BEHAVIORAL;
```