Conway's Game of Life
Using FPGA

Presented by Gerin Fajardo, Kevin Foyet,
Tyler Marts, and Raymond Wong
ECE 3300.02 - Digital Circuit Design Using Verilog
Professor Mohamed Aly

ABSTRACT

Conway's Game of Life is a zero-player game where a cellular automaton is played on a 2-dimensional square grid, created by mathematician John Horton Conway in 1970. On this grid, each cell (square) can occupy one of two states, alive or dead, with the general goal being the observation of how the grid of cells evolves as a whole.

Cells live or die based on four rules:

1) Any living cell with fewer than two neighbors dies.
2) Any living cell with more than three living neighbors dies.
3) Any living cell with two or three living neighbors may live unchanged.
4) Any deceased cell with exactly three living neighbors can come to life.

This project report will focus on recreating this game through the use of the Nexys A7 FPGA board, and the hardware description language (HDL) Verilog will be used to program it to follow the game's rules.

On the Nexys A7 board, eleven of the sixteen switches will be used as an 8-bit speed control, overflow control, enable, and reset. All buttons will be used to control the location of the cursor, as well as a toggle for a cell's state. Lastly, seven of the eight seven-segment display digits will be used to display the value of the speed and the current generation of the game. As for the visuals, a 16x16 grid will be displayed through a VGA interface on a connected monitor.

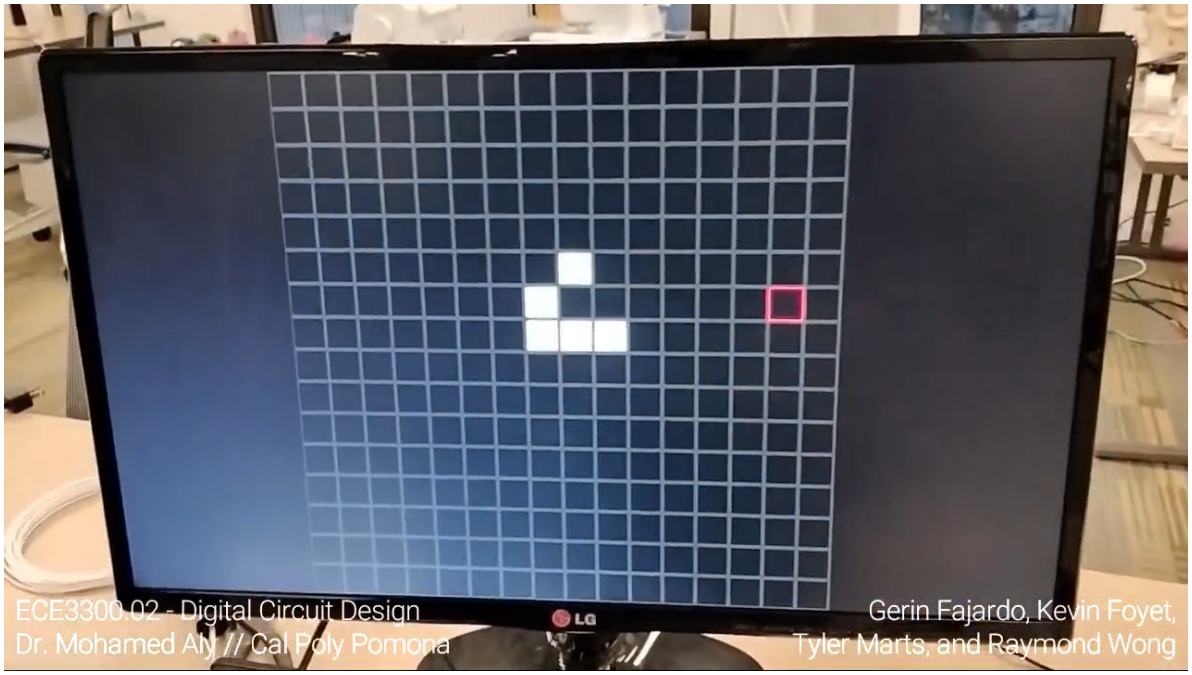


TABLE OF CONTENTS

OBJECTIVE

The objective of this project is to implement Mathematician's John Horton Conway's *Game of Life* simulation in Verilog, run it on an FPGA, and display the grid of cells via VGA. The development board used to develop this project was the Nexys A7 Artix-7 FPGA board.

DESIGN AND DESCRIPTION

DEVICES AND TOOLS USED

- Nexys A7 Artix-7 FPGA Trainer Board
- Xilinx Vivado 2018.1
- VGA-compatible Display

A. Nexys A7 Artix-7 FPGA Trainer Board

The Nexys A7 Artix-7 board is an entry-level development board designed purely for educational purposes. The board is typically used for prototyping and development in the realm of digital logic design and embedded systems. As the name suggests, the board's key component is the Xilinx Artix-7 FPGA (Field Programmable Gate Array) chip. This chip gives users of the board the ability to implement custom digital circuits through whatever HDL (Hardware Description Language) they choose (in this case, Verilog). The board provides a total of 45 I/O interfaces: 16 LEDs, 16 switches, 5 buttons, and 8 seven-segment displays which users can utilize in their projects. As for connectivity, multiple connection ports can be found on the board. These ports range from USB-UART, ethernet, micro SD cards, and even a VGA port to transmit visuals. Furthermore, the board utilizes several memory components like DDR3 RAM and non-volatile memory which prove useful when it comes to storing code and data for FPGA projects. For this project, an 8-bit speed control, overflow control, enable, and reset will be implemented onto 11 of the 16 switches on the board. To control the cursor's location on the grid and toggle a cell's state, all 5 buttons will be utilized. Lastly, to display the value of the speed in BCD and the current generation of the game in hexadecimal, 7 out of the 8 seven-segment displays will be used. A visual representation of this layout, a visual is provided in Figure 1.

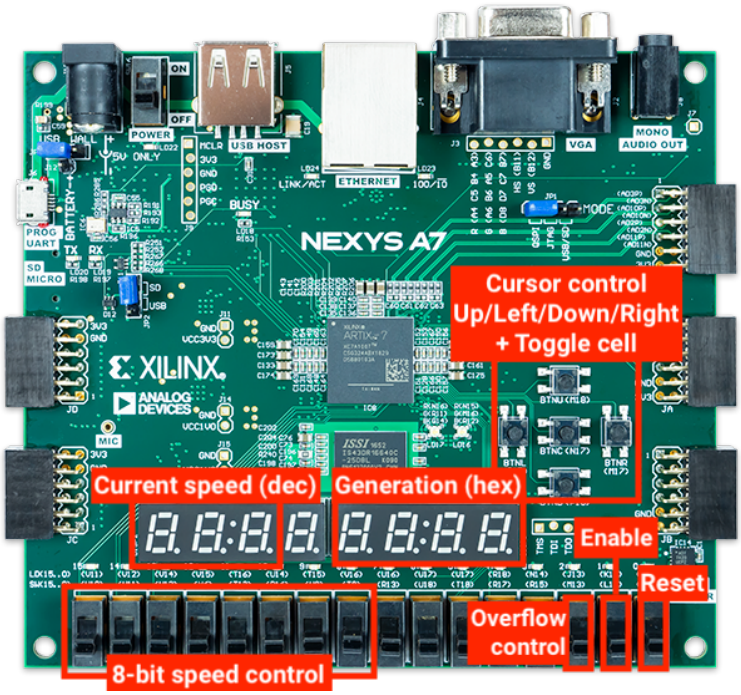


Figure 1: Nexys A7 100-T Board Layout

B. Xilinx Vivado 2018.1

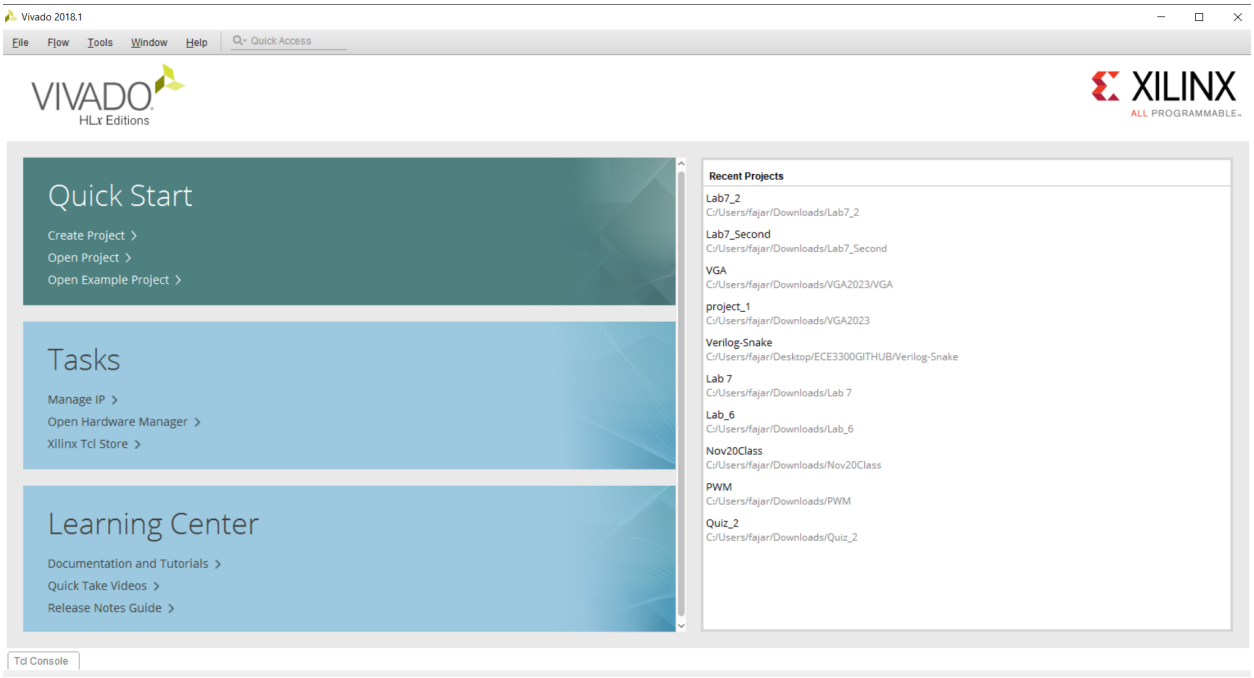


Figure 2: Xilinx Vivado 2018.1

Vivado is a software suite designed by Xilinx that is used to synthesize and analyze the designs of a hardware description language. Once synthesized, these designs can then be integrated and deployed onto FPGAs. Vivado itself is considered to be an Integrated Development Environment (IDE) and offers a wide variety of tools that can be used for FPGA designs. Various HDLs are supported on Vivardo such as Verilog, VHDL, and SystemVerilog. Lastly, with Vivado, users can see resource utilization, ease IP integration, and generate configuration files for FPGAs.

C. VGA-Compatible Display

A Video Graphics Array (VGA) monitor is a type of display that uses an analog interface in order to connect to other devices such as a computer. These monitors usually have a 15-pin VGA connector. It is through these connectors that analog video signals are transmitted in order to display images or videos on the screen with resolutions up to 640x480 pixels. Connecting an FPGA to a VGA monitor involves several steps, with the first being to generate a video signal in the VGA format. To do this, necessary timing signals must be created. These signals are horizontal sync (HSYNC), vertical sync (VSYNC), and the color RGB color information required for each pixel. The HSYNC signal increments on pixel clocks and controls the timing. The horizontal display time for the HSYNC signal is followed by a blanking time including a horizontal front porch, the horizontal sync pulse, and the horizontal back porch where at the end of the row, it will restart for the next. The same thing goes for the VSYNC signal where instead of horizontal, it's vertical with its corresponding front and back porch, and the sync pulse. A visual representation of this can be seen in Figure 4. The FPGA will output the video signal through the board's VGA port where it will utilize its digital-to-analog converters, converting the digital signals into a readable analog VGA signal that can be seen on the monitor.

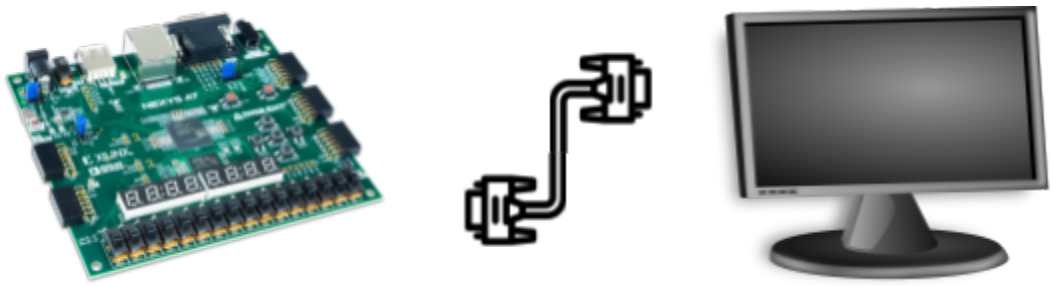


Figure 3: FPGA-VGA Setup for Displaying Grid

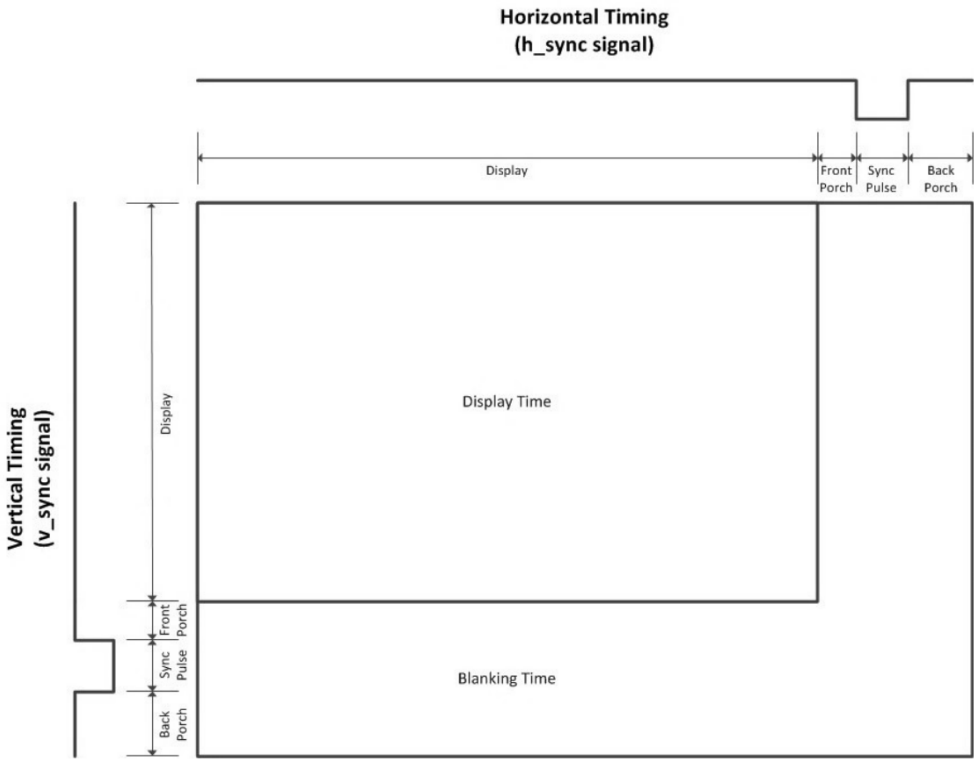


Figure 4: VGA Timing Diagram

RTL ANALYSIS ELABORATED DESIGN

Figure 5: RTL Schematic for *Conway's Game of Life*

VERILOG RTL CODE

Top Module

```
module Conway(
    input top_clk,
    input rst,
    input en,
    input overflow_ctrl,
    input [7:0] game_spd,
    input up_btn,
    input down_btn,
    input left_btn,
    input right_btn,
    input toggle_btn,

    // VGA outputs
    output reg [3:0] red,
    output reg [3:0] green,
    output reg [3:0] blue,
    output hsync,
    output vsync,

    // 7-segment display outputs
//    output [15:0] ssd_led,
    output [6:0] ssd_cc,
    output ssd_odp,
    output [7:0] ssd_an,

    output [15:0] debug_led
);
    reg [3:0] cursor_x = 4'b0000;
    reg [3:0] cursor_y = 4'b0000;
    wire [9:0] count_x;
    wire [9:0] count_y;
    wire displayArea;
    wire R, G, B;

    reg [4:0] last_input = 5'b11111; // { toggle, right, left, down, up }

    wire [11:0] speed_display; // BCD 0 - 255
    reg [15:0] gen_display = 16'b0000000000000000;
```

```
    assign debug_led = gen_display; // mostly for debug reasons, but it
looks cool

    reg [15:0] game_grid [0:15];
    reg [15:0] next_grid [0:15];
    reg [1:0] last_update = 2'b00;
    reg [1:0] request_update = 2'b00;

    wire ssd_clk;
    wire vga_clk;
    wire game_clk;

    clk_manager ssd_manager(
        .clk_in(top_clk),
        .clk_out(ssd_clk),
        .clk_spd(8'b00000000) // zero-out for sims
    );

    vga_clk_divider vga_divider(
        .clk(top_clk),
        .vga_clk(vga_clk)
    );

    clk_manager game_clk_mgr(
        .clk_in(top_clk),
        .clk_out(game_clk),
        .clk_spd(~game_spd) // inverted, that way higher number = faster
    );

    vga_generator vga_gen(
        .vga_clk(vga_clk),
        .hSync(hsync),
        .vSync(vsync),
        .countX(count_x),
        .countY(count_y),
        .displayArea(displayArea)
    );

    ssd_driver ssd_output (
        .ssd_clk(ssd_clk),
        .ssd_driver_port_inp({ speed_display, 4'b0000, gen_display }),
        .ssd_driver_port_en(8'b11101111),
```

```
        .ssd_driver_port_idp(1'b1),
        // outputs
//        .ssd_driver_port_led(ssd_led),
        .ssd_driver_port_cc(ssd_cc),
        .ssd_driver_port_odp(ssd_odp),
        .ssd_driver_port_an(ssd_an)
    );

    binary_bcd_converter binary_bcd(
        .bin(game_spd),
        .bcd(speed_display)
    );

    // grid update stuff
    reg [4:0] updater_x = 5'b00000;
    reg [4:0] updater_y = 5'b00000;
    reg [2:0] neighbors = 3'b000;


    always @ (posedge top_clk) begin: check_for_updates


        if (rst) begin
            for (updater_y = 0; updater_y < 16; updater_y = updater_y + 1)
begin
                for (updater_x = 0; updater_x < 16; updater_x = updater_x +
1) begin
                    game_grid[updater_y][updater_x] = 1'b0;
                    next_grid[updater_y][updater_x] = 1'b0; // also clear
this cell in next gen
                    gen_display = 0;
                end
            end
        end

        // if rst is false, check if grid needs to be updated
        else if ((last_update != request_update) & (game_spd > 0)) begin
            // first: update the NEXT generation
            for (updater_y = 0; updater_y < 16; updater_y = updater_y + 1)
begin
                for (updater_x = 0; updater_x < 16; updater_x = updater_x +
1) begin// the following IF statements sum up the neighbors of this cell
                    neighbors = 3'b000; // I know this CAN overflow, but
```

```
the outcome is the same for 0 and 8
                    if (overflow_ctrl) begin // disallow wrap-around
                        if (updater_y > 0) begin // top left, top mid, top
right
                            if ((updater_x > 0) &
game_grid[updater_y-1][updater_x-1]) neighbors = neighbors + 1;
                            if (game_grid[updater_y-1][updater_x])
neighbors = neighbors + 1;
                            if ((updater_x < 15) &
game_grid[updater_y-1][updater_x+1]) neighbors = neighbors + 1;
                        end
                        if (updater_y < 15) begin // bot left, bot mid, bot
right
                            if ((updater_x > 0) &
game_grid[updater_y+1][updater_x-1]) neighbors = neighbors + 1;
                            if (game_grid[updater_y+1][updater_x])
neighbors = neighbors + 1;
                            if ((updater_x < 15) &
game_grid[updater_y+1][updater_x+1]) neighbors = neighbors + 1;
                        end
                        // mid left, mid right
                        if (updater_x > 0 &
game_grid[updater_y][updater_x-1]) neighbors = neighbors + 1;
                        if (updater_x < 15 &
game_grid[updater_y][updater_x+1]) neighbors = neighbors + 1;
                    end
                    else begin // allow wrap-around
                        if
(game_grid[((updater_y-1)&5'b01111)][((updater_x-1)&5'b01111)]) neighbors =
neighbors + 1;
                        if (game_grid[((updater_y-1)&5'b01111)][((updater_x
)&5'b01111)]) neighbors = neighbors + 1;
                        if
(game_grid[((updater_y-1)&5'b01111)][((updater_x+1)&5'b01111)]) neighbors =
neighbors + 1;
                        if (game_grid[((updater_y
)&5'b01111)][((updater_x-1)&5'b01111)]) neighbors = neighbors + 1;
                        if (game_grid[((updater_y
)&5'b01111)][((updater_x+1)&5'b01111)]) neighbors = neighbors + 1;
                        if
(game_grid[((updater_y+1)&5'b01111)][((updater_x-1)&5'b01111)]) neighbors =
neighbors + 1;
                        if (game_grid[((updater_y+1)&5'b01111)][((updater_x
```

```
)&5'b01111)]) neighbors = neighbors + 1;
                        if
(game_grid[((updater_y+1)&5'b01111)][((updater_x+1)&5'b01111)]) neighbors =
neighbors + 1;
                    end


                    // rules: (n = neighbors)
                    // if (n < 2), die
                    // if (n == 2), stay dead or alive
                    // if (n == 3), become alive
                    // if (n > 3), die
                    if (neighbors == 2) next_grid[updater_y][updater_x] =
game_grid[updater_y][updater_x]; // stay
                    else if (neighbors == 3)
next_grid[updater_y][updater_x] = 1'b1; // alive
                    else next_grid[updater_y][updater_x] = 1'b0; // die

                end
            end


            // next: update the visual display
            for (updater_y = 0; updater_y < 16; updater_y = updater_y + 1)
begin
                for (updater_x = 0; updater_x < 16; updater_x = updater_x +
1) begin
                    game_grid[updater_y][updater_x] =
next_grid[updater_y][updater_x];
                end
            end

            // and apply changes
            gen_display = gen_display + 1;
            last_update = last_update + 1;
        end

        // cursor movement update
        if (up_btn & ~last_input[0])    cursor_y = cursor_y - 1;
        if (down_btn & ~last_input[1])  cursor_y = cursor_y + 1;
        if (left_btn & ~last_input[2])  cursor_x = cursor_x - 1;
        if (right_btn & ~last_input[3]) cursor_x = cursor_x + 1;
        if (toggle_btn & ~last_input[4] & ~rst)
```

```
game_grid[cursor_y][cursor_x] = ~game_grid[cursor_y][cursor_x];

        last_input <= { toggle_btn, right_btn, left_btn, down_btn, up_btn
};
    end

    always @ (posedge game_clk) begin: display_next_gen
        if (en) request_update = request_update + 1;
    end

    assign R = (displayArea & game_grid[count_y / 30][(count_x-80) / 30]);
    assign G = (displayArea & game_grid[count_y / 30][(count_x-80) / 30]);
    assign B = (displayArea & game_grid[count_y / 30][(count_x-80) / 30]);


    wire x_edge, y_edge;
    assign x_edge = ((count_x-80) % 30 < 2) | ((count_x-80) % 30 > 27);
    assign y_edge = (count_y % 30 < 2) | (count_y % 30 > 27);

    always @ (posedge vga_clk) begin: write_rgb_output
        if (count_x > 560 | count_x < 80) begin
            red <= 4'b0010;
            green <= 4'b0010;
            blue <= 4'b0010;
        end else if ((count_y/30 == cursor_y) & (count_x-80)/30 ==
cursor_x) begin
            red <=   ((x_edge | y_edge) ? 4'b1111 : {4{R}});
            green <= ((x_edge | y_edge) ? 4'b0000 : {4{G}});
            blue <=  ((x_edge | y_edge) ? 4'b0000 : {4{B}});
        end else begin
            red <=   ((x_edge | y_edge) ? 4'b0100 : {4{R}}); // 4'b0111 is
grey, used for drawing cell lines
            green <= ((x_edge | y_edge) ? 4'b0100 : {4{G}});
            blue <=  ((x_edge | y_edge) ? 4'b0100 : {4{B}});
        end
    end
```

CLK_Manager Module

```
module clk_manager (
    input clk_in,
```

```
    input [7:0] clk_spd,
    output clk_out
);

    reg toggle_clk = 0;
    reg [23:0] clk_counter = 24'b000000000000000000000000;

    always @ (posedge clk_in) begin
        if (clk_counter[23:16] >= clk_spd) begin
            toggle_clk = ~toggle_clk;
            clk_counter = 0;
        end else
            clk_counter = clk_counter + 1;
    end

    assign clk_out = toggle_clk;

endmodule
```

VGA_CLK_Divider Module

```
module vga_clk_divider(
    input clk,          // Input clock signal
    output reg vga_clk  // Output clock signal for VGA, declared as a
register because its value is updated in an always block
);
    integer a = 0;      // Counter variable used to track the number of
clock cycles

    // This always block triggers on the rising edge of the input clock
signal
    always @ (posedge clk) begin // If the counter 'a' is less than
'check', increment 'a' and keep the VGA clock low
        if(a < 3) begin
            a <= a + 1;     // Increment 'a' by 1
            vga_clk <= 0;   // Set VGA clock to 0 (low)
        end else begin // Once 'a' reaches the value of 'check', reset 'a'
and set VGA clock high
            a <= 0;         // Reset 'a' to 0
            vga_clk <= 1;   // Set VGA clock to 1 (high)
        end
```

```
    end
endmodule
```

VGA_Generator Module

```
module vga_generator(
    input vga_clk,

    output reg [9:0] countX,
    output reg [9:0] countY,
    output reg displayArea,
    output hSync,
    output vSync
 );

reg p_hSync;
reg p_vSync;

integer HFporch = 640; //Horizontal Front Porch
integer Hsync = 656; //Horizontal Sync
integer HBporch = 752; //Horizontal Back Porch
integer Hmax = 800; //Total Length of Line
integer VFporch = 480; //Vertical Front Porch
integer Vsync = 490; //Vertical Sync
integer VBporch = 492; //Vertical Back Porch
integer Vmax = 525; //Number of Rows

always @ (posedge vga_clk)
    begin
        if (countX == Hmax)
            countX <= 0;
        else
            countX <= countX + 1'b1;
    end

always @ (posedge vga_clk)
    begin
        if (countX == Hmax)
            begin
                if (countY == Vmax)
                    countY <= 0;
```

```
                else
                    countY <= countY + 1'b1;
            end
    end

always @ (posedge vga_clk)
    begin
        displayArea <= ((countX < HFporch) && (countY < VFporch));
    end

always @ (posedge vga_clk)
    begin
        p_hSync <= ((countX >= Hsync) && (countX < HBporch));
        p_vSync <= ((countY >= Vsync) && (countY < VBporch));
    end

assign vSync = ~p_vSync;
assign hSync = ~p_hSync;

endmodule
```

SSD_Driver (Seven-Segment-Display) Module

```
module ssd_driver(
        input ssd_clk,
        input [31:0] ssd_driver_port_inp,
        input [7:0] ssd_driver_port_en,
        input ssd_driver_port_idp,

        output [15:0] ssd_driver_port_led,
        output [6:0] ssd_driver_port_cc,
        output ssd_driver_port_odp,
        output [7:0] ssd_driver_port_an
    );

    reg [2:0] internal_toggle = 3'b000;
    reg [7:0] internal_anode = 8'b11111111;
    reg [3:0] internal_bcd = 4'b0000;

    assign ssd_driver_port_odp = ssd_driver_port_idp;
    assign ssd_driver_port_led = ssd_driver_port_inp[15:0];
```

```
reg[6:0] ssd_driver_tmp_cc;
integer lsb = 0;
integer msb = 0;

always @ (posedge ssd_clk) begin:SEG_ENC
     case (internal_toggle)
             3'b000: begin
                internal_anode <= 8'b01111111;
                internal_bcd <= ssd_driver_port_inp[3:0];
             end
             3'b001: begin
                internal_anode <= 8'b11111110;
                internal_bcd <= ssd_driver_port_inp[7:4];
             end
             3'b010: begin
                internal_anode <= 8'b11111101;
                internal_bcd <= ssd_driver_port_inp[11:8];
             end
             3'b011: begin
                internal_anode <= 8'b11111011;
                internal_bcd <= ssd_driver_port_inp[15:12];
             end
             3'b100: begin
                internal_anode <= 8'b11110111;
                internal_bcd <= ssd_driver_port_inp[19:16];
             end
             3'b101: begin
                internal_anode <= 8'b11101111;
                internal_bcd <= ssd_driver_port_inp[23:20];
             end
             3'b110: begin
                internal_anode <= 8'b11011111;
                internal_bcd <= ssd_driver_port_inp[27:24];
             end
             3'b111: begin
                internal_anode <= 8'b10111111;
                internal_bcd <= ssd_driver_port_inp[31:28];
             end
     endcase
     case (internal_bcd)
         4'h0: ssd_driver_tmp_cc = 7'b0000001;
         4'h1: ssd_driver_tmp_cc = 7'b1001111;
```

```
            4'h2: ssd_driver_tmp_cc = 7'b0010010;
            4'h3: ssd_driver_tmp_cc = 7'b0000110;
            4'h4: ssd_driver_tmp_cc = 7'b1001100;
            4'h5: ssd_driver_tmp_cc = 7'b0100100;
            4'h6: ssd_driver_tmp_cc = 7'b0100000;
            4'h7: ssd_driver_tmp_cc = 7'b0001111;
            4'h8: ssd_driver_tmp_cc = 7'b0000000;
            4'h9: ssd_driver_tmp_cc = 7'b0001100;
            4'hA: ssd_driver_tmp_cc = 7'b0001000;
            4'hB: ssd_driver_tmp_cc = 7'b1100000;
            4'hC: ssd_driver_tmp_cc = 7'b0110001;
            4'hD: ssd_driver_tmp_cc = 7'b1000010;
            4'hE: ssd_driver_tmp_cc = 7'b0110000;
            4'hF: ssd_driver_tmp_cc = 7'b0111000;
            default: ssd_driver_tmp_cc = 7'hZZ;
         endcase
        internal_toggle = internal_toggle + 1;
    end
    assign ssd_driver_port_an = internal_anode | ~ssd_driver_port_en;
    assign ssd_driver_port_cc = ssd_driver_tmp_cc;
   endmodule
```

Binary_BCD_Converter Module

```
module binary_bcd_converter(
    input [7:0] bin,
    output reg [11:0] bcd
);
    integer i;

    always @(bin) begin
        bcd=0;
        for (i=0;i<8;i=i+1) begin                          //Iterate once
for each bit in input number
            if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;          //If any
BCD digit is >= 5, add three
            if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
            if (bcd[11:8] >= 5) bcd[11:8] = bcd[11:8] + 3;
            bcd = {bcd[10:0], bin[7-i]};                   //Shift one bit,
and shift in proper bit from input
        end
    end
```

```
endmodule
```

VERILOG CONSTRAINT FILE

```
## This file is a general .xdc for the Nexys A7-100T

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { top_clk
}]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {
top_clk }];

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { rst }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { en }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports {
overflow_ctrl }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3]
}]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4]
}]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5]
}]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6]
}]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7]
}]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports {
game_spd[0] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports {
game_spd[1] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports {
game_spd[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports {
game_spd[3] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports {
game_spd[4] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports {
game_spd[5] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports {
game_spd[6] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports {
game_spd[7] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
```

```
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports {
debug_led[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]


##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[6]
}]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[5]
}]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[4]
}]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[3]
}]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[2]
}]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[1]
}]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { ssd_cc[0]
}]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { ssd_odp
}]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { ssd_an[0]
}]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { ssd_an[1]
}]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { ssd_an[2]
```

```
}]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { ssd_an[3]
}]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { ssd_an[4]
}]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { ssd_an[5]
}]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { ssd_an[6]
}]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { ssd_an[7]
}]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports {
toggle_btn }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { up_btn
}]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { left_btn
}]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { right_btn
}]; #IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { down_btn
}]; #IO_L9N_T1_DQS_D13_14 Sch=btnd


##VGA Connector
set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { red[0]
}]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { red[1]
}]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { red[2]
}]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { red[3]
}]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]
set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { green[0]
}]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { green[1]
}]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { green[2]
}]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { green[3]
}]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]
set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { blue[0]
}]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { blue[1]
}]; #IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { blue[2]
}]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { blue[3]
}]; #IO_L4P_T0_35 Sch=vga_b[3]
set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { hsync }];
#IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { vsync }];
#IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

VERILOG TESTBENCH CODE

Conway_TB Simulation Module

```
module Conway_TB;

    // Inputs
    reg top_clk;
    reg rst;
    reg en;
    reg overflow_ctrl;
    reg [7:0] game_spd;
    reg up_btn;
    reg down_btn;
    reg left_btn;
    reg right_btn;
    reg toggle_btn;

    // Outputs
    wire [3:0] red;
    wire [3:0] green;
    wire [3:0] blue;
    wire hsync;
    wire vsync;
    wire [6:0] ssd_cc;
    wire ssd_odp;
    wire [7:0] ssd_an;
    wire [15:0] debug_led;

    // Instantiate the Conway module
    Conway uut (
        .top_clk(top_clk),
        .rst(rst),
        .en(en),
        .overflow_ctrl(overflow_ctrl),
        .game_spd(game_spd),
        .up_btn(up_btn),
        .down_btn(down_btn),
        .left_btn(left_btn),
        .right_btn(right_btn),
        .toggle_btn(toggle_btn),
        .red(red),
```

```
        .green(green),
        .blue(blue),
        .hsync(hsync),
        .vsync(vsync),
        .ssd_cc(ssd_cc),
        .ssd_odp(ssd_odp),
        .ssd_an(ssd_an),
        .debug_led(debug_led)
    );

    // Clock initialization
    initial begin
        top_clk = 0;
        forever #5 top_clk = ~top_clk;
    end

    // Monitor output values
    always @(posedge top_clk) begin
        $display("Red: %b, Green: %b, Blue: %b, HSync: %b, VSync: %b", red,
green, blue, hsync, vsync);
    end

    // Testbench stimulus
    initial begin
        // Provide test values to inputs
        rst = 1;
        en = 0;
        overflow_ctrl = 0;
        game_spd = 8'd10; // Example game speed
        up_btn = 0;
        down_btn = 0;
        left_btn = 0;
        right_btn = 0;
        toggle_btn = 0;

        // Wait for a few clock cycles
        #100;

        // De-assert reset
        rst = 0;

        // Test scenario 1
        #50;
```

```
        en = 1;
        overflow_ctrl = 1;
        game_spd = 8'd5;
        up_btn = 1;
        #20;
        up_btn = 0;
        down_btn = 1;
        #20;
        down_btn = 0;
        left_btn = 1;
        #20;
        left_btn = 0;
        right_btn = 1;
        #20;
        right_btn = 0;
        toggle_btn = 1;
        #20;
        toggle_btn = 0;

        // Test scenario 2
        #50;
        en = 0;
        overflow_ctrl = 0;
        game_spd = 8'd3;
        up_btn = 0;
        down_btn = 1;
        #20;
        down_btn = 0;
        left_btn = 1;
        #20;
        left_btn = 0;
        right_btn = 0;
        toggle_btn = 1;
        #20;
        toggle_btn = 0;

        // Additional stimulus scenarios
        #50;
        en = 1;
        overflow_ctrl = 1;
        game_spd = 8'd7;
        left_btn = 1;
        #30;
```

```
        left_btn = 0;
        right_btn = 1;
        #40;
        right_btn = 0;
        toggle_btn = 1;
        #20;
        toggle_btn = 0;
        up_btn = 1;
        #30;
        up_btn = 0;
        down_btn = 1;
        #30;
        down_btn = 0;
        toggle_btn = 1;
        #20;
        toggle_btn = 0;

        // Add more stimulus cycles as needed

        // End simulation
        $finish;
    end

endmodule
```

SIMULATION WAVEFORM

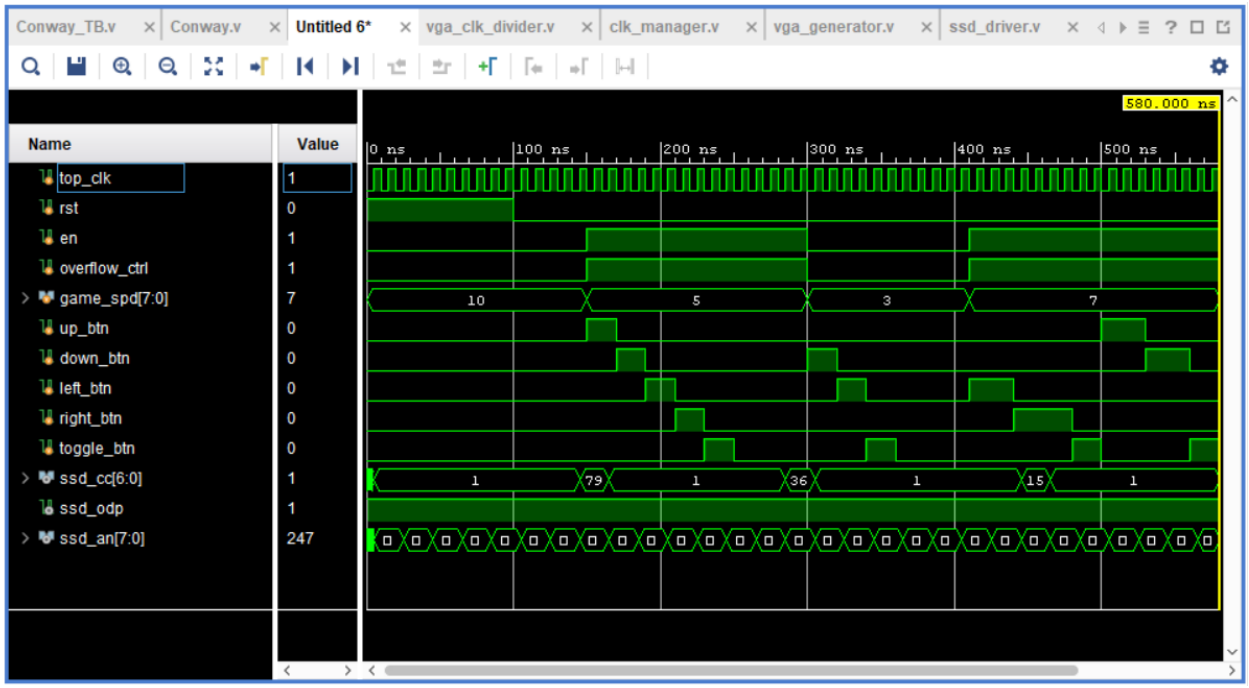
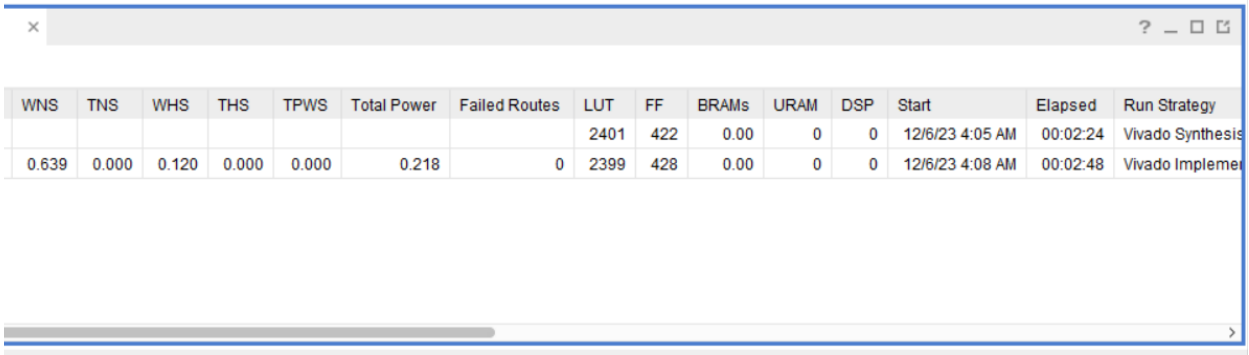


Figure 6: Testbench Simulation Waveform

SYNTHESIS REPORT

Upon synthesizing *Conway's Game of Life* in Vivado, certain data became available regarding resource consumption. Figure 7 gives a general breakdown as to what the total power being used is, and how many lookup tables (LUTs) and flip-flops (FFs) are used. For this project, 2401 LUTs and 422 FFs were used. Figure 8 further breaks down the total power used and the breakdown of where that power is going. In total, 0.218W was used, with 0.120W being dynamic and 0.097W being device static. The rest went into the other components. 4% of the power went into the clocks, 28% of the power went into the signals, 30% of the power went into logic, and 38% of the power went to I/O.



WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
							2401	422	0.00	0	0	12/6/23 4:05 AM	00:02:24	Vivado Synthesis
0.639	0.000	0.120	0.000	0.000	0.218	0	2399	428	0.00	0	0	12/6/23 4:08 AM	00:02:48	Vivado Implemer

Figure 7: Vivado Synthesis Report for *Conway's Game of Life*

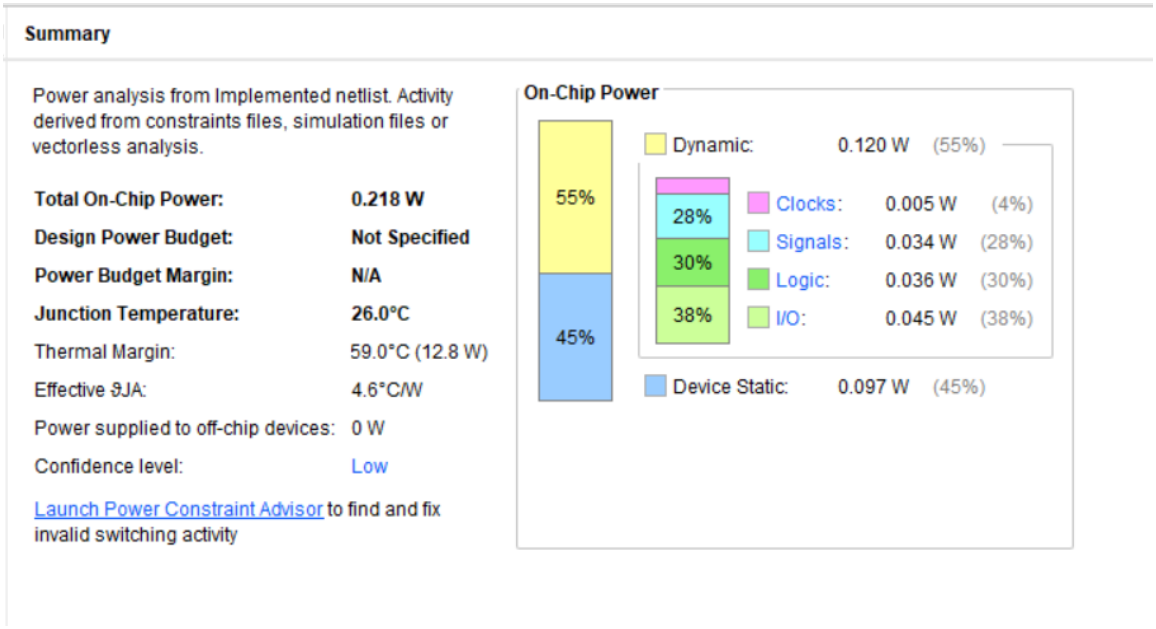


Figure 8: Vivado On-Chip Power Report for *Conway's Game of Life*

CONCLUSION

This project focused on recreating *Conway's Game of Life* on an FPGA board. To achieve such a goal, Verilog programming was used along with Xilinx's Vivado software to synthesize and implement the code on the FPGA. In order to emulate the game, a 16x16 grid was displayed on a VGA monitor. The FPGA board further emulated the game, being used for speed control, enable, reset, cursor location, cell toggle, speed, and current generation of the game. This was done by mapping these aspects to the board's switches, buttons, LEDs, and seven-segment display. In summary, this project utilized 2401 lookup tables, 422 flip-flops, and 0.218W of power. With the combination of Verilog knowledge, understanding of the Vivado software, and Nexys A7 Artix-7 board, the behavior of the game's cellular automaton was successfully simulated and recreated.

REFERENCES

"Binary to BCD and BCD to Binary." *RealDigital*, www.realdigital.org/doc/6dae6583570fd816d1d675b93578203d.

"Conway's Game of Life." *Wikipedia*, Wikimedia Foundation, 30 Nov. 2023, en.wikipedia.org/wiki/Conway%27s_Game_of_Life.

"Nexys A7-100T Master Constraints File." *GitHub*, Digilent, github.com/Digilent/digilent-xdc/blob/master/Nexys-A7-100T-Master.xdc.

Krishnajith S S. "Snake Game on FPGA in Verilog." *PDF*, slideshare, 2017, www.slideshare.net/sskrishnajith/snake-game-on-fpga-in-verilog.

Larson, Scott. "VGA Controller (VHDL)." *DigiKey*, 17 Mar. 2021, forum.digikey.com/t/vga-controller-vhdl/12794.