

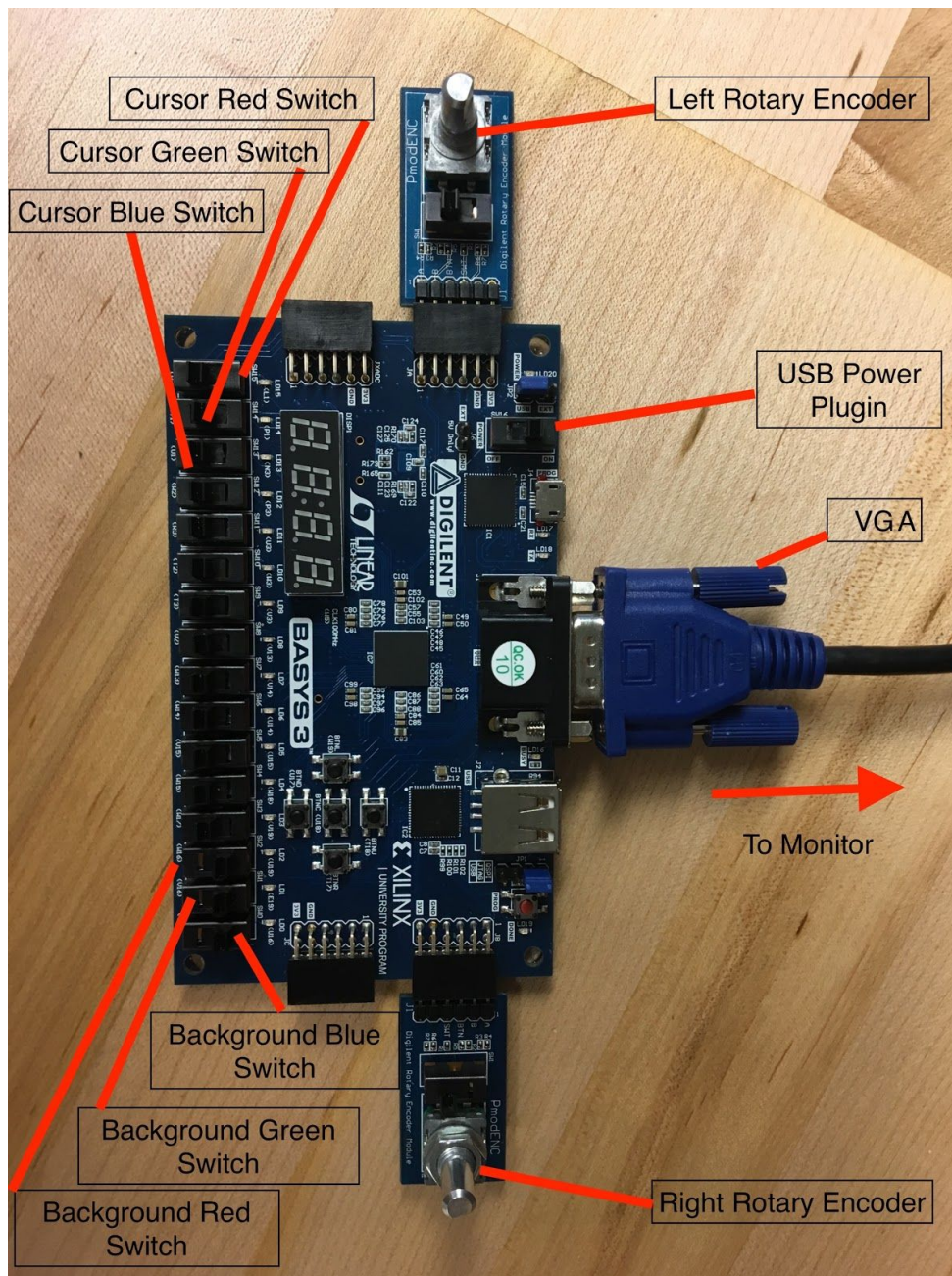
7. The Appendix

Table of Contents

Appendix A: Front Panel	2
Appendix B: Block Diagrams	3
Appendix C: State Diagrams	8
Appendix D: VHDL Code	10
Appendix E: Resource Utilization	29
Appendix F: Residual Warnings	30
Appendix G: Parts List	31
Appendix H: Waveform Graphs	32

Appendix A: Front Panel

Figure 1: Front of the FPGA Board: All of the different project components are labelled on the board and referred to by name in the report.

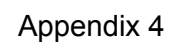


Appendix B: Block Diagrams

Figure 1: FPGA Inputs and Outputs: Top level block diagram for the system including the input and output signal flows through the FPGA unit

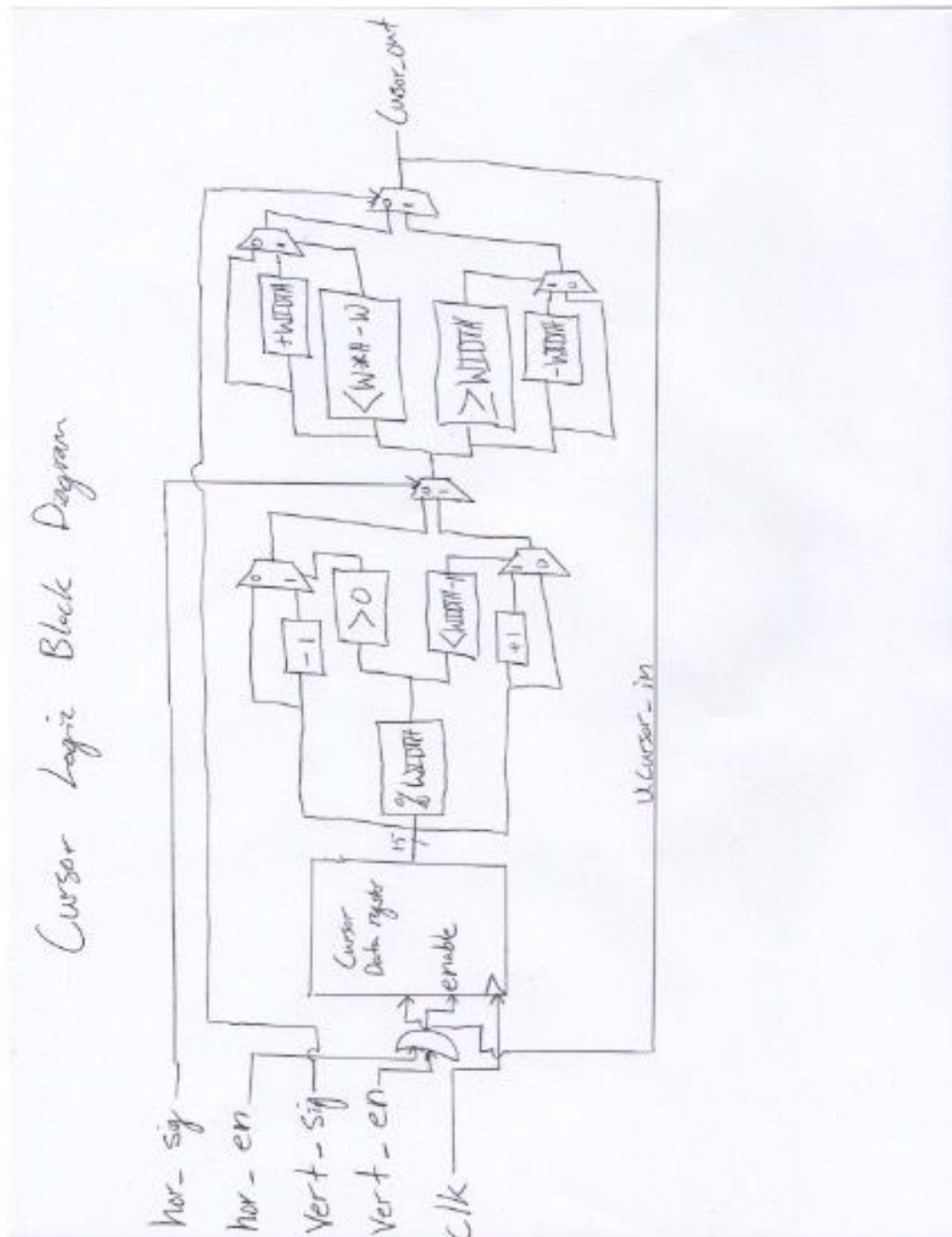


Figure 2: FPGA Shell Block Diagram: This describes the top level shell block diagram, including all major subcomponents, such as debouncers for inputs, encoder modules, Block RAM, and logic for clearing, pen up/down, and colors.



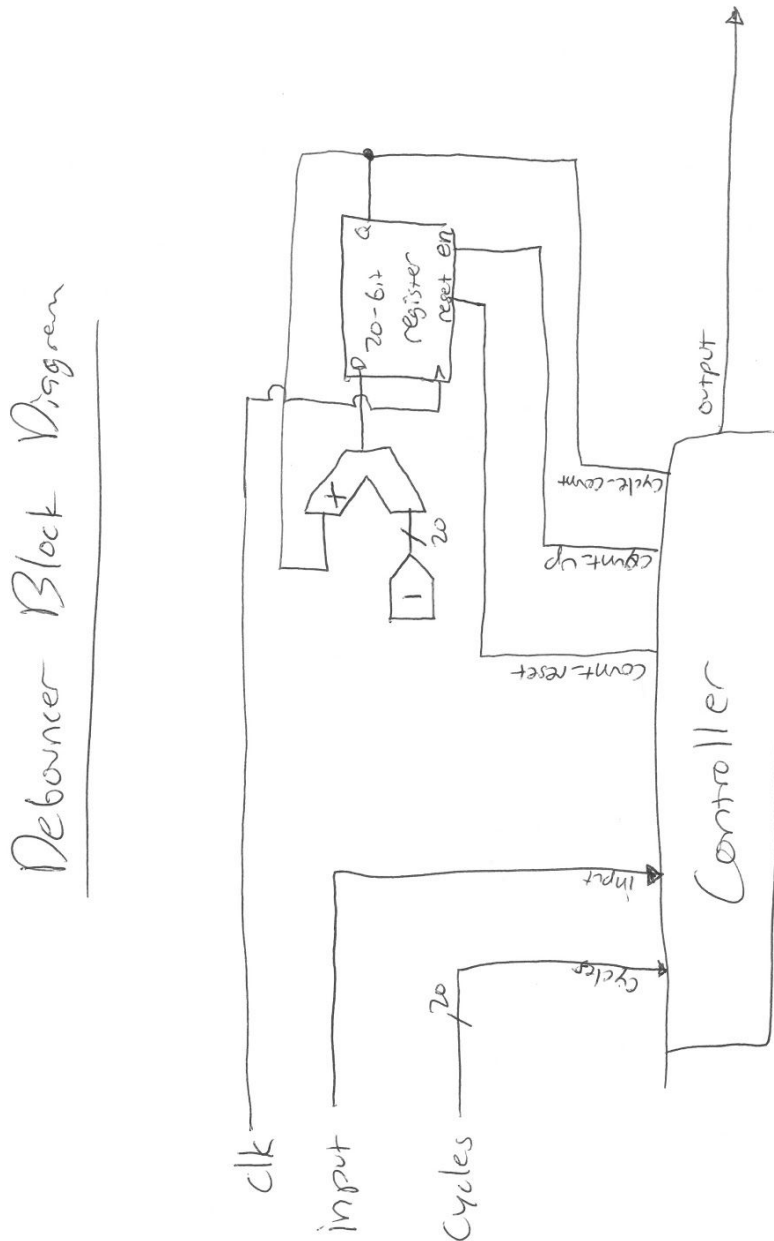
Appendix B: Block Diagrams

Figure 3: Cursor Logic Data Diagram: Logic flow through the cursor logic module including out of bounds checking as well as incrementing and decrementing updates to the cursor position



Appendix B: Block Diagrams

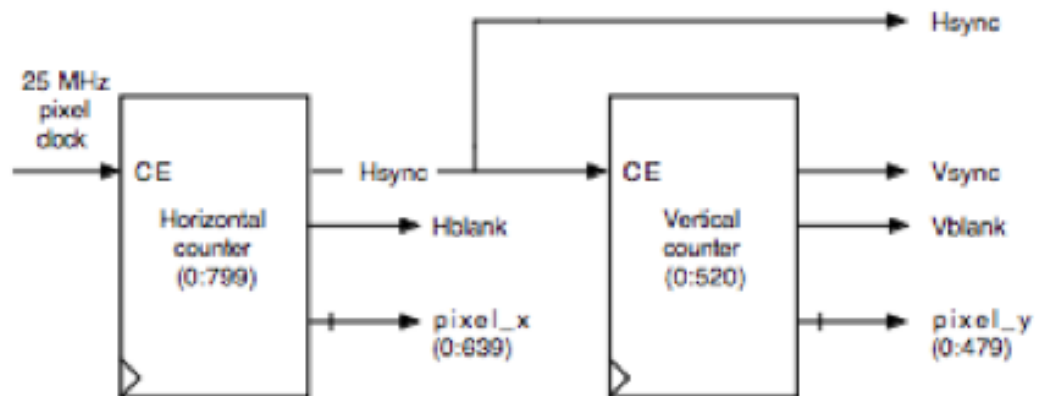
Figure 4: Debouncer Data Diagram: Block diagram for debouncer component showing the mechanism for the incrementation of cycle_count based on control signals count_up and count_reset



Appendix B: Block Diagrams

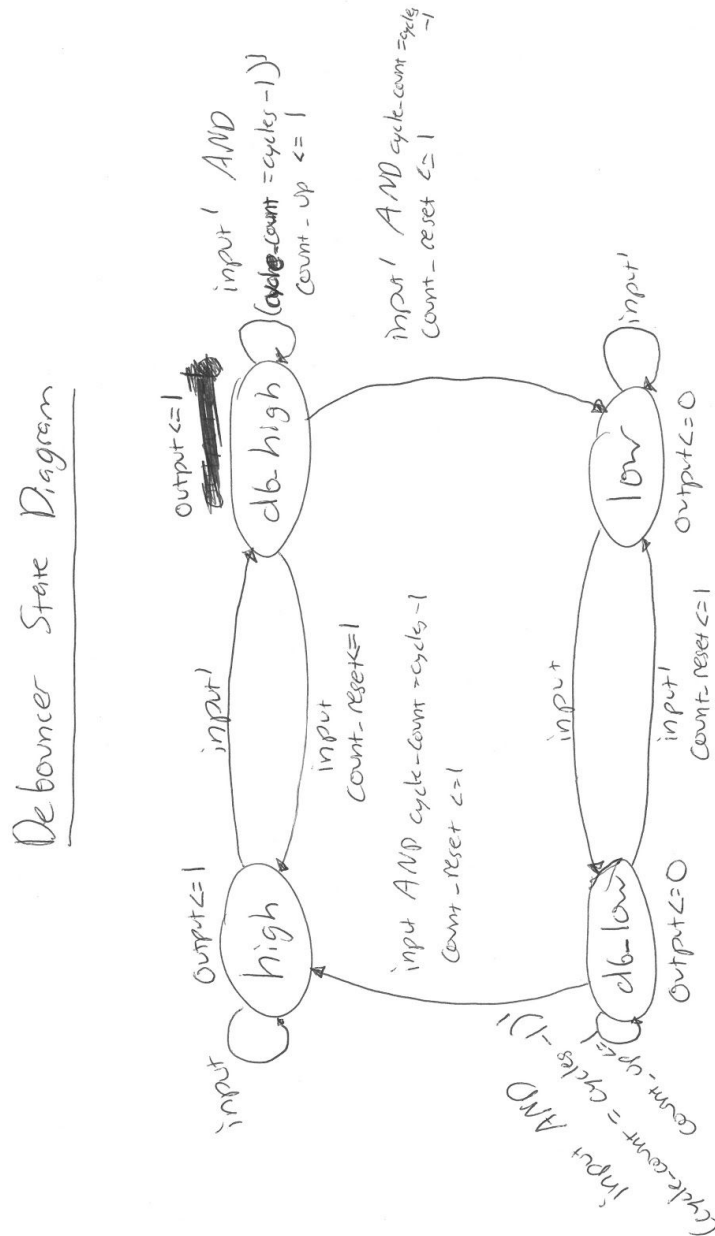
Figure 5: VGA Data diagram: Block diagram for the VGA component taken from the lecture slides, which shows the VGA handling and outputting data

VGA controller



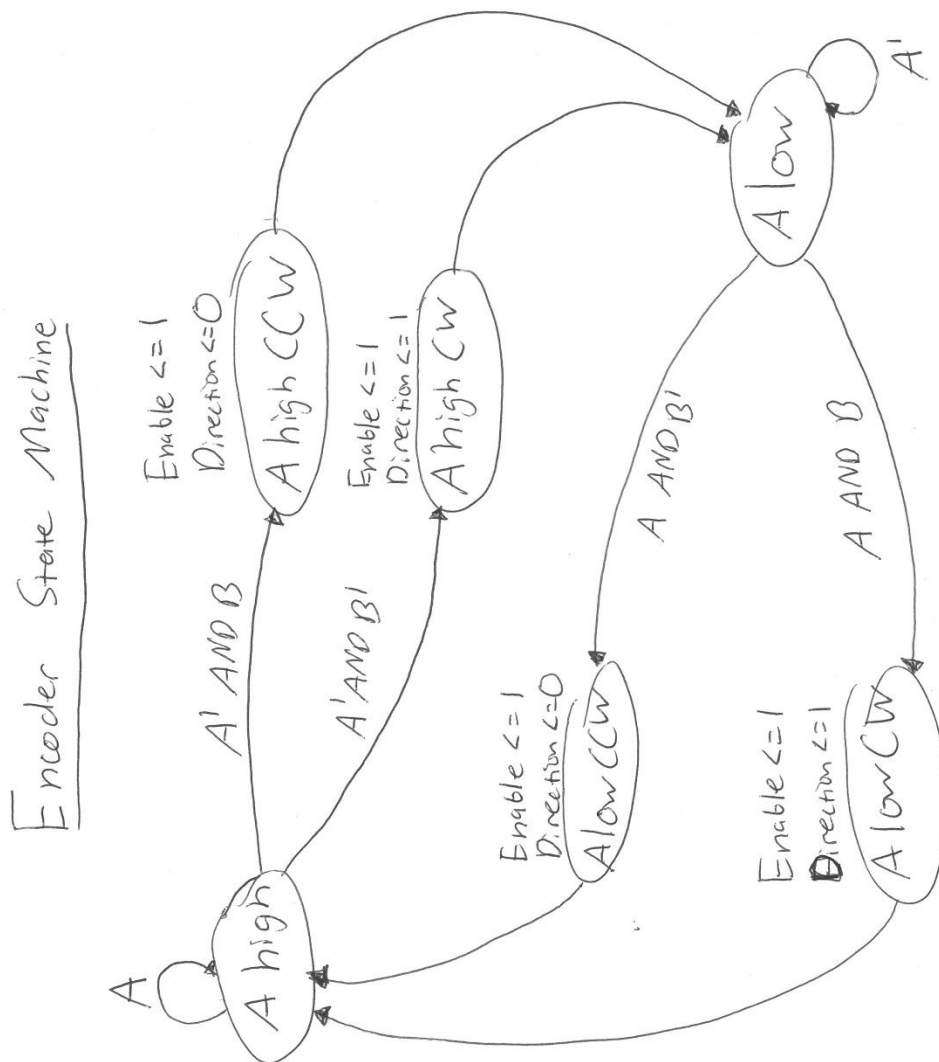
Appendix C: State Diagrams

Figure 1: Debouncer: State diagram logic for Debouncer component including debouncing the input signal to create the output signal when the number of clock cycles have elapsed while high or low.



Appendix C: State Diagrams

Figure 2: Signal Encoder: State diagram logic for Encoder component; rotating encoder results in enable being asserted with every change in A, with direction asserted simultaneously only if the direction is clockwise.



Appendix D: VHDL Code

Figure 1: Top Level Shell: Code for the top level shell, complete with entities, signals, port mapping, and auxiliary helper processes

```
-----
-- Company:                Engs 31 18X
-- Engineer:                Liam Jolley / Graham Hazlett
--
-- Create Date:            08/15/18
-- Design Name:
-- Module Name:            top_level_shell
-- Project Name:           Etch-a-Sketch
-- Target Devices:         Digilent Basys3 (Artix 7)
-- Tool versions:          Vivado 2016.1
-- Description:            Top Level Shell connecting all parts of the Etch-a-Sketch as well as initiating processes
--                          to link the code base together
--
-- Dependencies:           cursor_logic, encoder_logic, debouncer, and VGA
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.math_real.all;          -- needed for arithmetic

library UNISIM;                  -- needed for the BUFG component
use UNISIM.Vcomponents.ALL;

entity top_level_shell is
port (
    clk: in STD_LOGIC;
    hor_a: in STD_LOGIC; -- left rotary encoder data
    hor_b: in STD_LOGIC; -- left rotary encoder data
    vert_a: in STD_LOGIC; -- right rotary encoder data
    vert_b: in STD_LOGIC; -- right rotary encoder data
    Vsync: out STD_LOGIC; -- vga data
    Hsync: out STD_LOGIC; -- vga data
    pixel_data: out STD_LOGIC_VECTOR(11 downto 0); -- final pixel color data
    switch_red: in STD_LOGIC; -- red component of cursor color
    switch_green: in STD_LOGIC; -- green component of cursor color
    switch_blue: in STD_LOGIC; -- blue component of cursor color

```

```

        background_red: in STD_LOGIC; -- red component of background color
        background_green: in STD_LOGIC; -- green component of background color
        background_blue: in STD_LOGIC; -- blue component of background color
        clear: in STD_LOGIC; -- clear initiating data
        up_down: in STD_LOGIC -- cursor up/down initiating data
    );
end top_level_shell;

architecture Behavioral of top_level_shell is

-- COMPONENT DECLARATIONS
-- Cursor register
component cursor_logic is
    Port ( clk: in STD_LOGIC;
          hor_sig: in STD_LOGIC; -- 1 is clockwise (right), 0 counterclockwise (left)
          vert_sig: in STD_LOGIC; -- 1 is clockwise (up), 0 counterclockwise (down)
          hor_en: in STD_LOGIC; -- left knob being turned or not
          vert_en: in STD_LOGIC; -- right knob being turned or not
          cursor_out: out STD_LOGIC_VECTOR(14 downto 0) ); -- current cursor location
end component;

-- Rotary encoder
component encoder_logic is
    Port ( A: in std_logic; -- feedback data from turn
          B: in std_logic; -- feedback data from turn
          clk: in std_logic;
          Enable: out std_logic; -- 1 means has detected turn, 0 no turn detected this tick
          Direction: out std_logic -- 1 is clockwise, 0 counterclockwise
    );
end component;

component debouncer is
    Port ( clk: in std_logic;
          input: in std_logic;
          output: out std_logic;
          cycles: in std_logic_vector(19 downto 0) -- debouncing time
    );
end component;

-- Block RAM
COMPONENT Block_Ram
PORT (
    clka : IN STD_LOGIC;
    ena : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNT0 0); -- write enable
    addra : IN STD_LOGIC_VECTOR(14 DOWNT0 0); -- write address
    dina : IN STD_LOGIC_VECTOR(3 DOWNT0 0); -- data written
    clkb : IN STD_LOGIC;

```

```

    enb : IN STD_LOGIC;
    addrb : IN STD_LOGIC_VECTOR(14 DOWNT0 0); -- read address
    doutb : OUT STD_LOGIC_VECTOR(3 DOWNT0 0) -- data read
  );
END COMPONENT;

-- VGA
Component VGA is
  port ( vclk : in STD_LOGIC; -- 25 MHz clock that VGA runs on
        Vsync: out STD_LOGIC;
        Hsync: out STD_LOGIC;
        video_on: out STD_LOGIC;
        pixel_x: out std_logic_vector(9 downto 0); -- x coordinate of VGA
        pixel_y: out std_logic_vector(9 downto 0)); -- y coordinate of VGA
end Component;

-----
-- SIGNAL DECLARATIONS
-- Signals for the serial clock divider, which divides the 100 MHz clock down to 50 MHz
constant SCLK_DIVIDER_VALUE: integer := 2;
constant COUNT_LEN: integer := integer(ceil( log2( real(SCLK_DIVIDER_VALUE) ) ));
signal sclkdir: unsigned(COUNT_LEN-1 downto 0) := (others => '0'); -- clock divider counter
signal sclk_unbuf: std_logic := '0'; -- unbuffered serial clock
signal sclk: std_logic := '0'; -- internal serial clock
signal sample_counter: unsigned(16 downto 0) := (others => '0');

constant WIDTH: unsigned(7 downto 0) := x"a0"; --160
constant HEIGHT: unsigned(7 downto 0) := x"78"; --120

-- intermediate signals
signal hor_en_s: std_logic := '0'; -- intermediate signal for the horizontal enable
signal hor_sig_s: std_logic := '0'; -- intermediate signal for the horizontal direction
signal vert_en_s: std_logic := '0'; -- intermediate signal for the vertical enable
signal vert_sig_s: std_logic := '0'; -- intermediate signal for the vertical direction
signal cursor_s: std_logic_vector(14 downto 0) := (others => '0'); -- intermediate signal for the cursor position
coming from cursor logic
signal ram_read_addr: std_logic_vector(14 downto 0) := (others => '0'); -- signal used to feed into ram read address
signal pixel_x: std_logic_vector(9 downto 0); -- intermediate signal for pixel x location
signal pixel_y: std_logic_vector(9 downto 0); -- intermediate signal for pixel y location
signal combined_position: std_logic_vector(14 downto 0) := (others => '0'); -- full position
signal video_on: std_logic := '0'; -- vga video on signal
signal raw_color: std_logic_vector(3 downto 0); -- raw color data coming from RAM.
signal color_in: std_logic_vector(3 downto 0); -- 3 bit rgb color to be fed into RAM
signal background_color: std_logic_vector(2 downto 0) := "111"; -- signal for background color, initialized as white.
signal mp_curr, mp_prev: std_logic := '0'; -- signals for monopulser
signal db_up_down, mp_up_down: std_logic := '0'; -- cursor up or down

-- write enable variables

```

```

signal write_en_s: std_logic_vector(0 downto 0) := "1"; -- final write_enable signal
signal vsync_s: std_logic := '0'; -- vsync signal
signal vsync_first: std_logic := '0'; -- to hold previous vsync signal
signal vsync_second: std_logic := '0'; -- to hold new vsync signal

--debouncing variables
constant DB_CYCLES: std_logic_vector(19 downto 0) := X"0ffff"; --20000
signal db_hor_a, db_hor_b, db_vert_a, db_vert_b: std_logic; -- debounced values

-----
begin

vsync <= vsync_s; -- updatating vsync
combined_position <= std_logic_vector(resize(unsigned(pixel_y(8 downto 2)) * WIDTH + unsigned(pixel_x(9
downto 2)),15)); -- converting coordnates to RAM data value

-- Clock buffer for sclk
-- The BUFG component puts the signal onto the FPGA clocking network
Slow_clock_buffer: BUFG
    port map (I => sclk_unbuf,
              O => sclk );

-- Divide the 100 MHz clock down to 50 MHz, then toggling a flip flop gives the final
-- 25 MHz system clock
Serial_clock_divider: process(clk)
begin
    if rising_edge(clk) then
        if sclkdiv = SCLK_DIVIDER_VALUE-1 then
            sclkdiv <= (others => '0');
            sclk_unbuf <= NOT(sclk_unbuf);
        else
            sclkdiv <= sclkdiv + 1;
        end if;
    end if;
end process Serial_clock_divider;

-- process to handle clearing
clear_logic: process(clk, clear, combined_position, switch_red, switch_green, switch_blue)
begin
    if clear = '1' and unsigned(pixel_x) / 4 < WIDTH and unsigned(pixel_y) / 4 < HEIGHT then
        ram_read_addr <= combined_position;
        color_in <= "0000";
    else
        color_in <= switch_red & switch_green & switch_blue & '1'; --last bit set as '1' to denote it is not background
        ram_read_addr <= cursor_s;
    end if;
end process clear_logic;

```

```

-- process to monopulse the debounced up_down signal
Monopulser: process(clk, db_up_down)
begin
    if rising_edge(clk) then
        mp_curr <= db_up_down;
        mp_prev <= mp_curr;
    end if;
    mp_up_down <= mp_curr and not(mp_prev);
end process Monopulser;

-- process to handle picking the pen up or putting it down
Up_down_logic: process(clk, db_up_down)
begin
    if rising_edge(clk) then
        if mp_up_down = '1' then
            write_en_s <= not(write_en_s);
        end if;
    end if;
end process Up_down_logic;

-- process to change the background color
switch_color_processor: process( background_red, background_green, background_blue)
begin
    background_color <= background_red & background_green & background_blue;
end process switch_color_processor;

--combinatorial process to process 4 bit pixel data for 12 bit vga
vga_color_processor: process(combined_position, raw_color)
begin
    if video_on = '0' then
        pixel_data <= X"000";
    elsif raw_color(0) = '0' then -- if this has never been written to; ie is background color
        pixel_data <= background_color(2) & background_color(2) & background_color(2) & background_color(2) &
            background_color(1) & background_color(1) & background_color(1) & background_color(1) &
            background_color(0) & background_color(0) & background_color(0) & background_color(0);
    else
        pixel_data <= raw_color(3) & raw_color(3) & raw_color(3) & raw_color(3) &
            raw_color(2) & raw_color(2) & raw_color(2) & raw_color(2) &
            raw_color(1) & raw_color(1) & raw_color(1) & raw_color(1);
    end if;
end process vga_color_processor;

---- Process to assert the write enable if necessary, only needed if you want
---- to update only at bottom of the screen. This slows the drawing down, so
---- we chose to keep it commented out in the end.
--write_en_check: process(clk)
--begin

```



```

-- if rising_edge(clk) then
--   vsync_second <= vsync_first;
--   vsync_first <= vsync_s;
--   if vsync_first = '0' and vsync_second = '1' then
--     write_en_s <= "1";
--   else
--     write_en_s <= "0";
--   end if;
-- end if;
--end process write_en_check;

-- Instantiate the horizontal rotary encoder
Hor_Encoder: encoder_logic
  PORT MAP (
    A => db_hor_a,
    B => db_hor_b,
    clk => clk,
    Enable => hor_en_s,
    Direction => hor_sig_s );

-- Instantiate the vertical rotary encoder
Vert_Encoder: encoder_logic
  PORT MAP (
    A => db_vert_a,
    B => db_vert_b,
    clk => clk,
    Enable => vert_en_s,
    Direction => vert_sig_s );

-- Instantiate debouncers for encoders / pen up-down:
Hor_A_Debouncer: debouncer
  PORT MAP (
    clk => clk,
    input => hor_a,
    output => db_hor_a,
    cycles => DB_CYCLES );

Hor_B_Debouncer: debouncer
  PORT MAP (
    clk => clk,
    input => hor_b,
    output => db_hor_b,
    cycles => DB_CYCLES );

Vert_A_Debouncer: debouncer
  PORT MAP (
    clk => clk,
    input => vert_a,

```

```

    output => db_vert_a,
    cycles => DB_CYCLES );

Vert_B_Debouncer: debouncer
PORT MAP (
    clk => clk,
    input => vert_b,
    output => db_vert_b,
    cycles => DB_CYCLES );

Up_Down_Debouncer: debouncer
PORT MAP (
    clk => clk,
    input => up_down,
    output => db_up_down,
    cycles => DB_CYCLES );

-- Instantiate the cursor update
Cursor_Update: cursor_logic
PORT MAP (
    clk => clk,
    hor_sig => hor_sig_s,
    vert_sig => vert_sig_s,
    hor_en => hor_en_s,
    vert_en => vert_en_s,
    cursor_out => cursor_s
);

-- Instantiate the RAM
RAM : Block_Ram
PORT MAP (
    clka => clk,
    ena => '1',
    wea => write_en_s, -- signal triggered when the pen is down
    addra => ram_read_addr, -- current cursor position
    dina => color_in, -- mark current position as the desired color
    clkb => clk,
    enb => '1',
    addrb => combined_position,
    doutb => raw_color
);

-- Instantiate the VGA
display: VGA port map(
    vclk => sclk,
    Vsync => Vsync_s,
    Hsync => Hsync,
    video_on => video_on,
    -- runs on the 25 MHz clock

```

```
        pixel_x => pixel_x,  
        pixel_y => pixel_y);  
  
end Behavioral;
```

Appendix D: VHDL Code

Figure 2: Cursor Logic: Cursor logic code complete with inputs and outputs as well as the cursor update process and unsigned variable holding the cursor location in the module

```
-----  
-- Company: ENGS31  
-- Engineer: Liam Jolley  
--  
-- Create Date: 08/14/2018 0  
-- Design Name:  
-- Module Name: cursor_logic - Behavioral  
-- Project Name: Etch-a-Sketch  
-- Target Devices:  
-- Tool Versions:  
-- Description: Logic to update the cursor location  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
  
entity cursor_logic is  
  Port ( clk: in STD_LOGIC;  
        hor_sig: in STD_LOGIC; -- 1 for clockwise and 0 for counterclockwise  
        vert_sig: in STD_LOGIC; -- 1 for clockwise and 0 for counterclockwise  
        hor_en: in STD_LOGIC;  
        vert_en: in STD_LOGIC;  
        cursor_out: out STD_LOGIC_VECTOR(14 downto 0) );  
end cursor_logic;  
  
architecture Behavioral of cursor_logic is  
  
  constant WIDTH : integer:= 160;
```

```

constant HEIGHT : integer:= 120;
signal ucursor_in: unsigned(14 downto 0) := "001000000000000"; --random start location, change to middle

begin

update_cursor: process(clk, hor_en, vert_en)
begin
    if rising_edge(clk) then
        if hor_en = '1' then -- horizontal knob turning
            if hor_sig = '0' then -- counter-clockwise
                if (ucursor_in mod WIDTH) > 0 then -- check to see if in first column
                    ucursor_in <= ucursor_in - 1; -- move left
                end if;
            else -- clockwise
                if (ucursor_in mod WIDTH) < WIDTH-1 then -- check to see if in last column
                    ucursor_in <= ucursor_in + 1; -- move right
                end if;
            end if;
        end if;
        if vert_en = '1' then -- vertical knob turning
            if vert_sig = '0' then --counter-clockwise
                if ucursor_in < ((WIDTH * HEIGHT) - WIDTH) then -- check to see if in bottom row
                    ucursor_in <= ucursor_in + WIDTH; -- move up
                end if;
            else -- clockwise
                if ucursor_in >= (WIDTH) then -- check to see if in top row
                    ucursor_in <= ucursor_in - WIDTH; -- move down
                end if;
            end if;
        end if;
        cursor_out <= std_logic_vector(ucursor_in); -- update cursor
    end if;
end process update_cursor;

end Behavioral;

```

Appendix D: VHDL Code

Figure 3: Debouncer: Code for the debounce module, which can take in a generic input, and assert the output only if the input has been asserted for the desired number of clock cycles specified by the 'cycles' input.

```
-----  
-- Company: ENGS31  
-- Engineer: Graham Hazlett  
--  
-- Create Date: 08/20/2018 12:05:24 PM  
-- Design Name:  
-- Module Name: debouncer - Behavioral  
-- Project Name: Etch-a-Sketch  
-- Target Devices: Diligent Baysis3 (Atrix 7)  
-- Tool Versions: Vivado 2016.1  
-- Description: Module to debounce a generic signal for the number of clock cycles  
--              specified by the 'cycles' input.  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.numeric_std.ALL;  
use IEEE.math_real.all;           -- needed for arithmetic  
  
-- Uncomment the following library declaration if using  
-- arithmetic functions with Signed or Unsigned values  
--use IEEE.NUMERIC_STD.ALL;  
  
-- Uncomment the following library declaration if instantiating  
-- any Xilinx leaf cells in this code.  
--library UNISIM;  
--use UNISIM.VComponents.all;  
  
entity debouncer is  
    Port ( clk: in std_logic;  
          input: in std_logic;
```

```

        output: out std_logic;
        cycles: in std_logic_vector(19 downto 0)
    );
end debouncer;

```

architecture Behavioral of debouncer is

```

type db_state_type is (high,db_high,low,db_low);
signal curr_state, next_state: db_state_type := low;
signal cycle_count: unsigned(19 downto 0) := (others => '0');
signal count_up, count_reset: std_logic := '0';

```

```
begin
```

```
-- combinatorial logic denoting state changes based on inputs:
```

```
debounce: process(input, cycle_count, cycles, curr_state)
```

```
begin
```

```
    count_up <= '0';
```

```
    count_reset <= '0';
```

```
    case curr_state is
```

```
        when high =>
```

```
            output <= '1';
```

```
            if input = '0' then
```

```
                next_state <= db_high;
```

```
            else
```

```
                next_state <= high;
```

```
            end if;
```

```
        when db_high =>
```

```
            output <= '1';
```

```
            if input = '0' then
```

```
                if unsigned(cycle_count) = unsigned(cycles) - 1 then
```

```
                    count_reset <= '1';
```

```
                    next_state <= low;
```

```
                else
```

```
                    count_up <= '1';
```

```
                    next_state <= db_high;
```

```
                end if;
```

```
            else
```

```
                count_reset <= '1';
```

```
                next_state <= high;
```

```
            end if;
```

```
        when low =>
```

```
            output <= '0';
```

```
            if input = '1' then
```

```
                next_state <= db_low;
```

```
            else
```

```
                next_state <= low;
```



```

        end if;
    when db_low =>
        output <= '0';
        if input = '1' then
            if unsigned(cycle_count) = unsigned(cycles) - 1 then
                count_reset <= '1';
                next_state <= high;
            else
                count_up <= '1';
                next_state <= db_low;
            end if;
        else
            count_reset <= '1';
            next_state <= low;
        end if;
    end case;
end process debounce;

-- synchronous process updating count based on count_up and count_reset
count_update: process(clk, count_up, count_reset)
begin
    if rising_edge(clk) then
        if count_up = '1' then
            cycle_count <= cycle_count + 1;
        elsif count_reset = '1' then
            cycle_count <= (others => '0');
        end if;
    end if;
end process count_update;

-- state update process
debounce_state_update: process(clk)
begin
    if rising_edge(clk) then
        curr_state <= next_state;
    end if;
end process debounce_state_update;

end Behavioral;

```

Appendix D: VHDL Code

Figure 4: Encoder: Code for the encoder module, which takes in (preferably debounced) A and B signals from a rotary encoder, and outputs 'enable' for one clock cycle when it detects a rotation, and 'direction' simultaneously if it is in the clockwise direction.

```
-----
-- Company: ENGS31
-- Engineer: Graham Hazlett
--
-- Create Date: 08/14/2018 03:41:31 PM
-- Design Name:
-- Module Name: encoder_logic - Behavioral
-- Project Name: Etch-a-Sketch
-- Target Devices: Diligent Baysis3 (Atrix 7)
-- Tool Versions: Vivado 2016.1
-- Description: Module to interpret encoder signals, and output signals denoting
-- a rotation, and the direction of that rotation
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity encoder_logic is
  Port ( A: in std_logic;
        B: in std_logic;
```

```

    clk: in std_logic;
    Enable: out std_logic; -- 1 means has detected turn, 0 no turn detected this tick
    Direction: out std_logic -- 1 is clockwise, 0 counterclockwise
);
end encoder_logic;

```

architecture Behavioral of encoder_logic is

```

type state_type is (Ahigh, Alow, AhighCW, AhighCCW, AlowCW, AlowCCW);
signal curr_state, next_state: state_type := Alow;

```

```

begin

```

```

-- state update process
state_update: process(clk)
begin
    if rising_edge(clk) then
        curr_state <= next_state;
    end if;
end process;

```

```

-- combinatorial process detnoting state changes and output assertions

```

```

main: process(clk, A, B)

```

```

begin

```

```

    Enable <= '0';
    Direction <= '0';
    next_state <= curr_state;
    case curr_state is

```

```

        when Ahigh =>
            if A = '0' then
                if B = '0' then
                    next_state <= AhighCW;
                else
                    next_state <= AhighCCW;
                end if;
            end if;

```

```

        when AhighCCW =>
            Enable <= '1';
            Direction <= '0';
            next_state <= Alow;

```

```

        when AhighCW =>
            Enable <= '1';
            Direction <= '1';
            next_state <= Alow;

```

```

when Alow =>
  if A = '1' then
    if B = '0' then
      next_state <= AlowCCW;
    else
      next_state <= AlowCW;
    end if;
  end if;

when AlowCCW =>
  Enable <= '1';
  Direction <= '0';
  next_state <= Ahigh;

when AlowCW =>
  Enable <= '1';
  Direction <= '1';
  next_state <= Ahigh;

end case;
end process;

end Behavioral;

```

Appendix D: VHDL Code

Figure 5: VGA: This VGA code is mostly the work of Professor Hansen with a few additions from Liam. The code handles the clock input and scans through the display.

```
-- E.W. Hansen, Engs 31 / CoSc 56 18X
-- Edited by Liam Jolley
-- Day 23 design exercise: VGA Controller

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
ENTITY VGA IS
    PORT ( vclk : in STD_LOGIC; --25 MHz clock
          Vsync: out STD_LOGIC;
          Hsync: out STD_LOGIC;
          video_on: out STD_LOGIC;
          pixel_x: out std_logic_vector(9 downto 0);
          pixel_y: out std_logic_vector(9 downto 0));
end VGA;

architecture behavior of VGA is

    signal H_video_on : STD_LOGIC := '0';
    signal V_video_on : STD_LOGIC := '0';
    --Add your signals here
    signal x : unsigned(9 downto 0) := (others=>'0');
    signal y : unsigned(9 downto 0) := to_unsigned(479, 10);
    signal V_enable : std_logic;
    --VGA Constants (taken directly from VGA Class Notes)
    constant left_border : integer := 48; -- back porch
    constant h_display : integer := 640; -- active image
    constant right_border : integer := 16; -- front porch
    constant h_retrace : integer := 96; -- Hsync pulse width
    constant HSCAN : integer := left_border + h_display
    + right_border + h_retrace; -- =800
    constant top_border : integer := 29; -- back porch
    constant v_display : integer := 480; -- active image
    constant bottom_border : integer := 10; -- front porch
    constant v_retrace : integer := 2; -- Vsync pulse width
    constant VSCAN : integer := top_border + v_display
    + bottom_border + v_retrace; -- =521

BEGIN
```

```

pixel_x <= std_logic_vector(x);
pixel_y <= std_logic_vector(y);

-- Hsync generating process
Hsync_proc : process(vclk, x)
begin
    if rising_edge(vclk) then
        if x < HSCAN-1 then -- horizontal counter
            x <= x+1;
        else
            x <= (others=>'0');
        end if;
    end if;

    if (x>=h_display) and (x<HSCAN)
        then H_video_on <= '0';
    else H_video_on <= '1';
    end if;

    if (x>=h_display+right_border) and (x<HSCAN-left_border)
        then Hsync <= '0';
    else Hsync <= '1';
    end if;

    if (x=HSCAN-left_border)
        then V_enable <= '1';
    else V_enable <= '0';
    end if;
end process Hsync_proc;

-- Vsync generating process
Vsync_proc : process(vclk, y) -- vertical counter
begin
    if rising_edge(vclk) then
        if V_enable = '1' then
            if y<VSCAN-1
                then y <= y+1;
            else y <= (others=>'0');
            end if;
        end if;
    end if;

    if (y>=v_display) and (y<VSCAN)
        then V_video_on <= '0';
    else V_video_on <= '1';
    end if;

    if (y>=v_display+bottom_border) and (y<VSCAN-top_border)

```

```
        then Vsync <= '0';  
        else Vsync <= '1';  
        end if;  
end process Vsync_proc;
```

```
-- Only enable video out , when H_video_out and V_video_out are high.  
-- It's important to set the output to zero when you aren't actively displaying video.  
-- That's how the monitor determines the black level.  
video_on <= '1' when (H_video_on = '1') and (V_video_on = '1') else '0';  
  
end behavior;
```














Appendix E: Resource Utilization

Figure 1: Resource Utilization Report: This table shows how many of each resource each module in the hierarchy uses to complete its process.

Name ¹	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
▼ top_level_shell	255	177	125	255	45	2.5	27	2
▶ Cursor_Update (curso...	103	30	49	103	8	0	0	0
▶ display (VGA)	54	20	25	54	12	0	0	0
▶ Hor_A_Debouncer (de...	11	22	8	11	2	0	0	0
▶ Hor_B_Debouncer (de...	11	22	8	11	2	0	0	0
▶ Hor_Encoder (encoder...	8	3	5	8	3	0	0	0
▼ RAM (Block_Ram)	16	6	9	16	0	2.5	0	0
> ▶ U0 (Block_Ram_blk...	16	6	9	16	0	2.5	0	0
▶ Up_Down_Debouncer ...	12	22	9	12	2	0	0	0
▶ Vert_A_Debouncer (de...	11	22	8	11	2	0	0	0
▶ Vert_B_Debouncer (de...	11	22	8	11	2	0	0	0
▶ Vert_Encoder (encoder...	16	3	9	16	3	0	0	0

Appendix F: Residual Warnings

Figure 1: Warnings Report: This screenshot shows all of the warnings thrown during the bitstream generation process. We've come to the conclusion that none of these warnings are critical because all deal with either extraneous variable declarations or sensitivity list warnings that are not vital to the program's execution, and did not impair functionality.

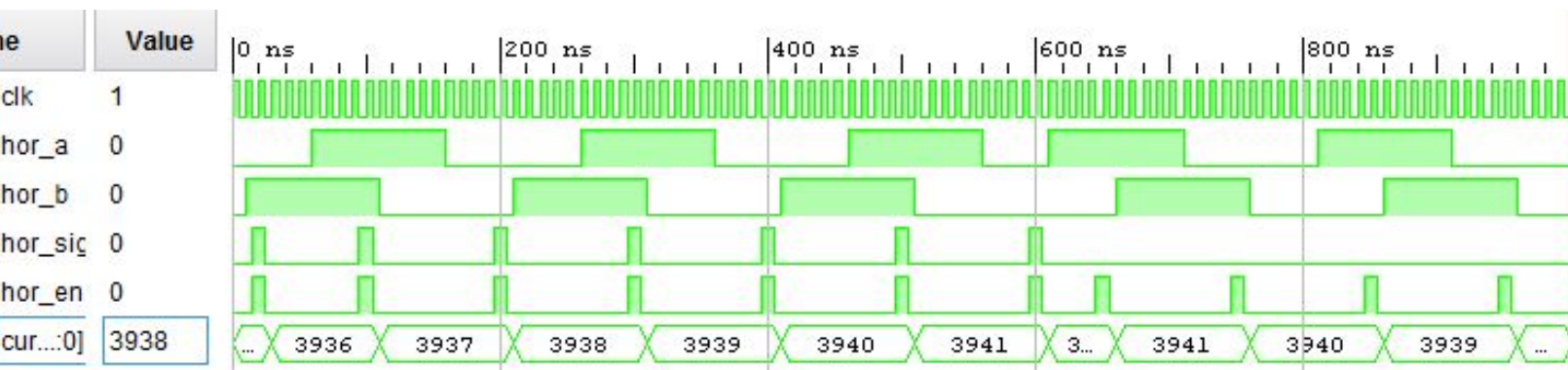
- ▼  Synthesis (20 warnings)
 - ▼  synth_1 (20 warnings)
 - ▼  [Synth 8-614] signal 'pixel_x' is read in the process but is not in the sensitivity list [top_level_shell.vhd:172] (7 more like this)
 -  [Synth 8-614] signal 'pixel_y' is read in the process but is not in the sensitivity list [top_level_shell.vhd:172]
 -  [Synth 8-614] signal 'cursor_s' is read in the process but is not in the sensitivity list [top_level_shell.vhd:172]
 -  [Synth 8-614] signal 'mp_curr' is read in the process but is not in the sensitivity list [top_level_shell.vhd:184]
 -  [Synth 8-614] signal 'mp_prev' is read in the process but is not in the sensitivity list [top_level_shell.vhd:184]
 -  [Synth 8-614] signal 'video_on' is read in the process but is not in the sensitivity list [top_level_shell.vhd:210]
 -  [Synth 8-614] signal 'background_color' is read in the process but is not in the sensitivity list [top_level_shell.vhd:210]
 -  [Synth 8-614] signal 'curr_state' is read in the process but is not in the sensitivity list [encoder_logic.vhd:58]
 - ▼  [Synth 8-6014] Unused sequential element cycle_count_reg was removed. [debouncer.vhd:109] (11 more like this)
 -  [Synth 8-6014] Unused sequential element ucursor_in_reg was removed. [cursor_logic.vhd:52]
 -  [Synth 8-6014] Unused sequential element x_reg was removed. [VGA.vhd:39]
 -  [Synth 8-6014] Unused sequential element y_reg was removed. [VGA.vhd:40]
 -  [Synth 8-6014] Unused sequential element Hor_A_Debouncer/cycle_count_reg was removed. [debouncer.vhd:109]
 -  [Synth 8-6014] Unused sequential element Hor_B_Debouncer/cycle_count_reg was removed. [debouncer.vhd:109]
 -  [Synth 8-6014] Unused sequential element Vert_A_Debouncer/cycle_count_reg was removed. [debouncer.vhd:109]
 -  [Synth 8-6014] Unused sequential element Vert_B_Debouncer/cycle_count_reg was removed. [debouncer.vhd:109]
 -  [Synth 8-6014] Unused sequential element Up_Down_Debouncer/cycle_count_reg was removed. [debouncer.vhd:109]
 -  [Synth 8-6014] Unused sequential element Cursor_Update/ucursor_in_reg was removed. [cursor_logic.vhd:52]
 -  [Synth 8-6014] Unused sequential element display/x_reg was removed. [VGA.vhd:39]
 -  [Synth 8-6014] Unused sequential element display/v_reg was removed. [VGA.vhd:40]

Appendix G: Parts List

- 1 FPGA Board
- 2 Rotary Encoders
- 1 VGA Cord
- 1 Monitor

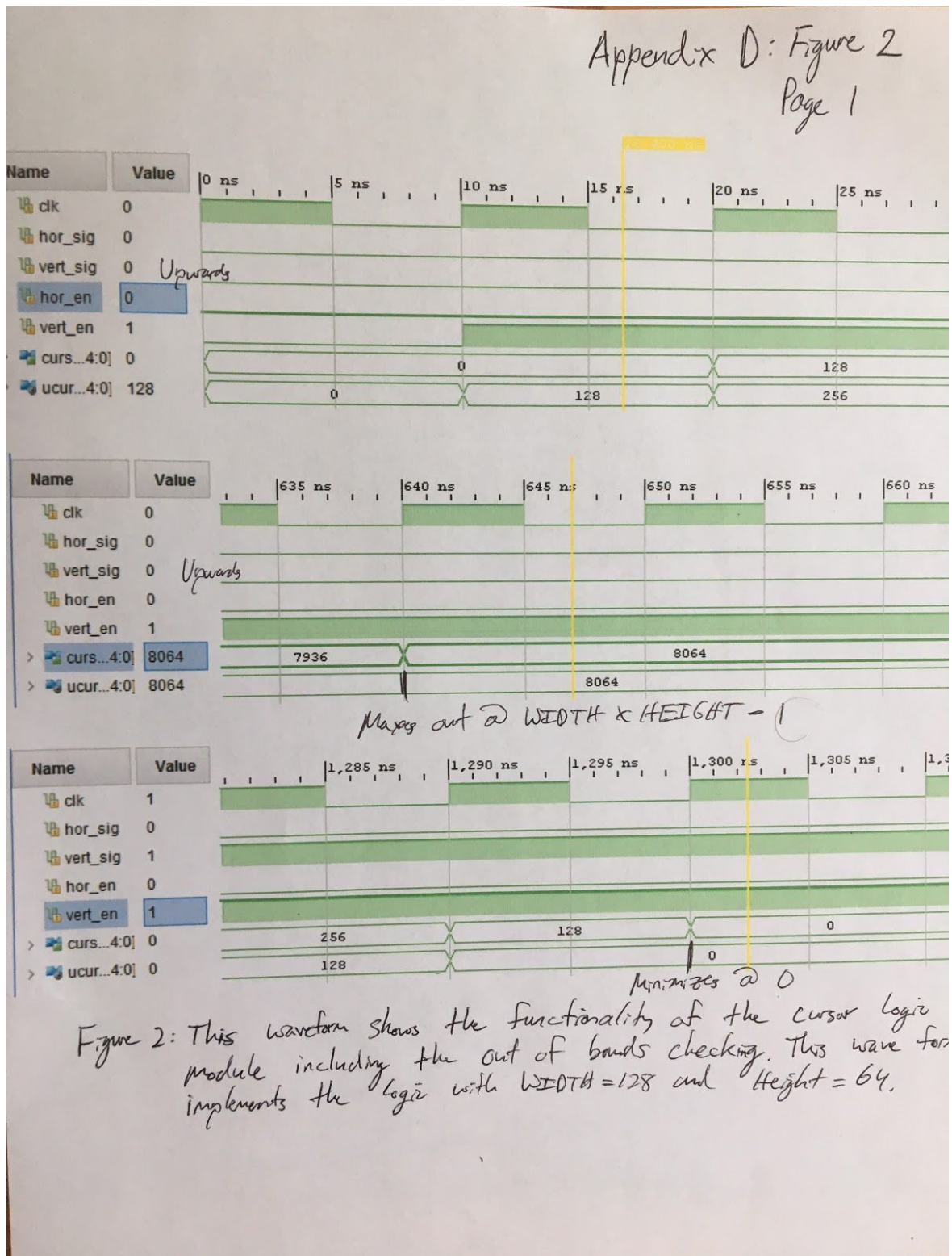
Appendix H: Waveform Graphs

Figure 1: Top Level Shell Waveform: The most important aspect of this waveform is that the rotary encoder signals are translated into cursor movement as shown in this figure.

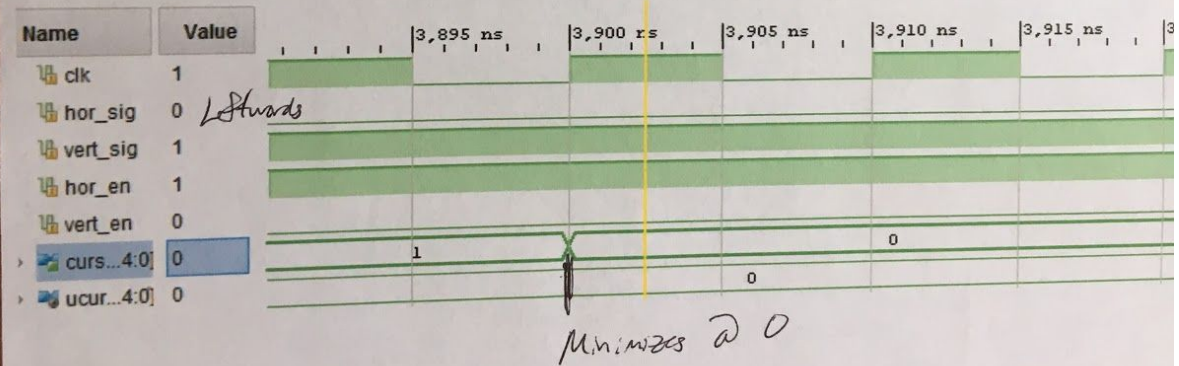
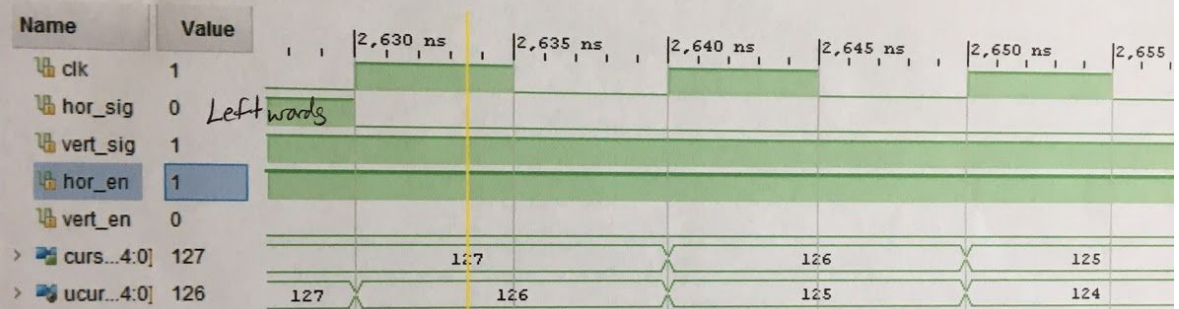
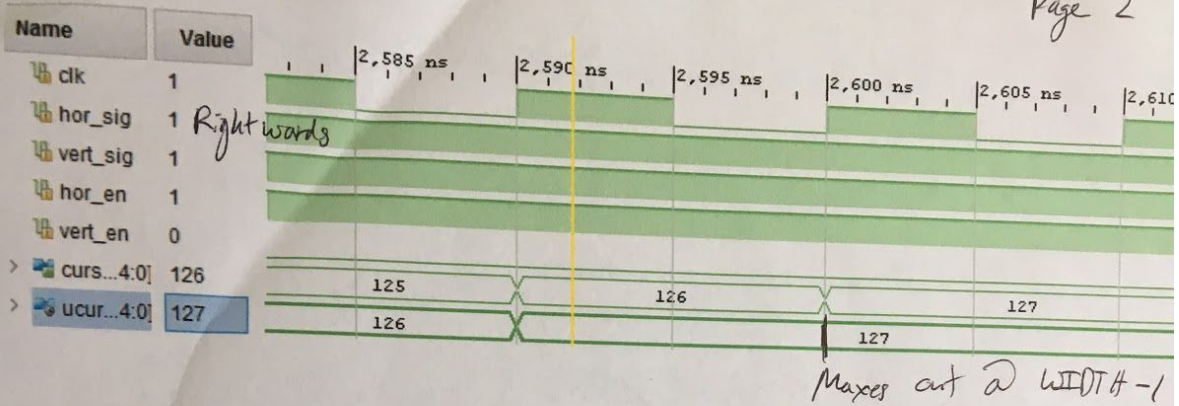


Appendix H: Waveform Graphs

Figure 2: Cursor Logic: See written description for information

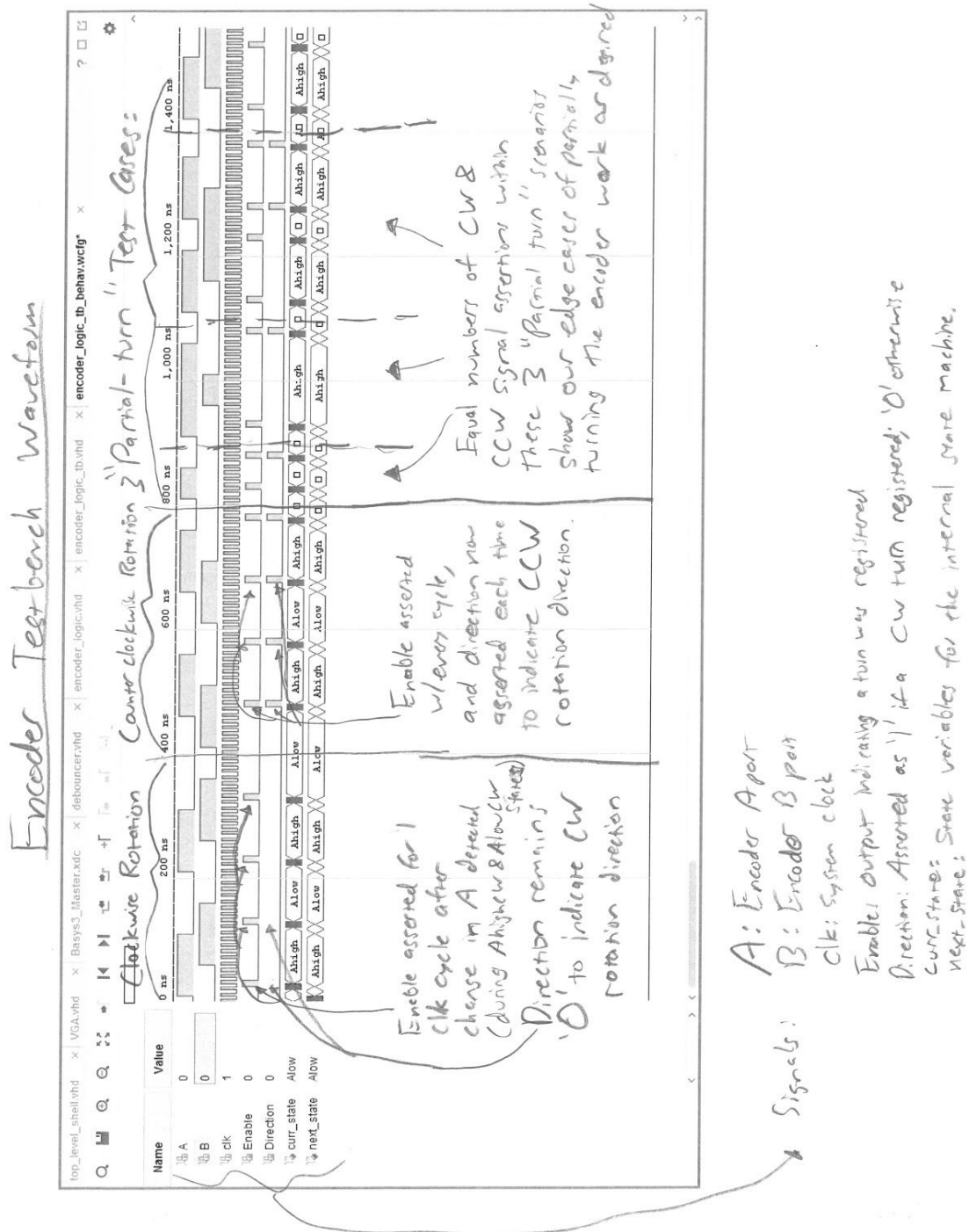


Appendix D: Figure 2
Page 2



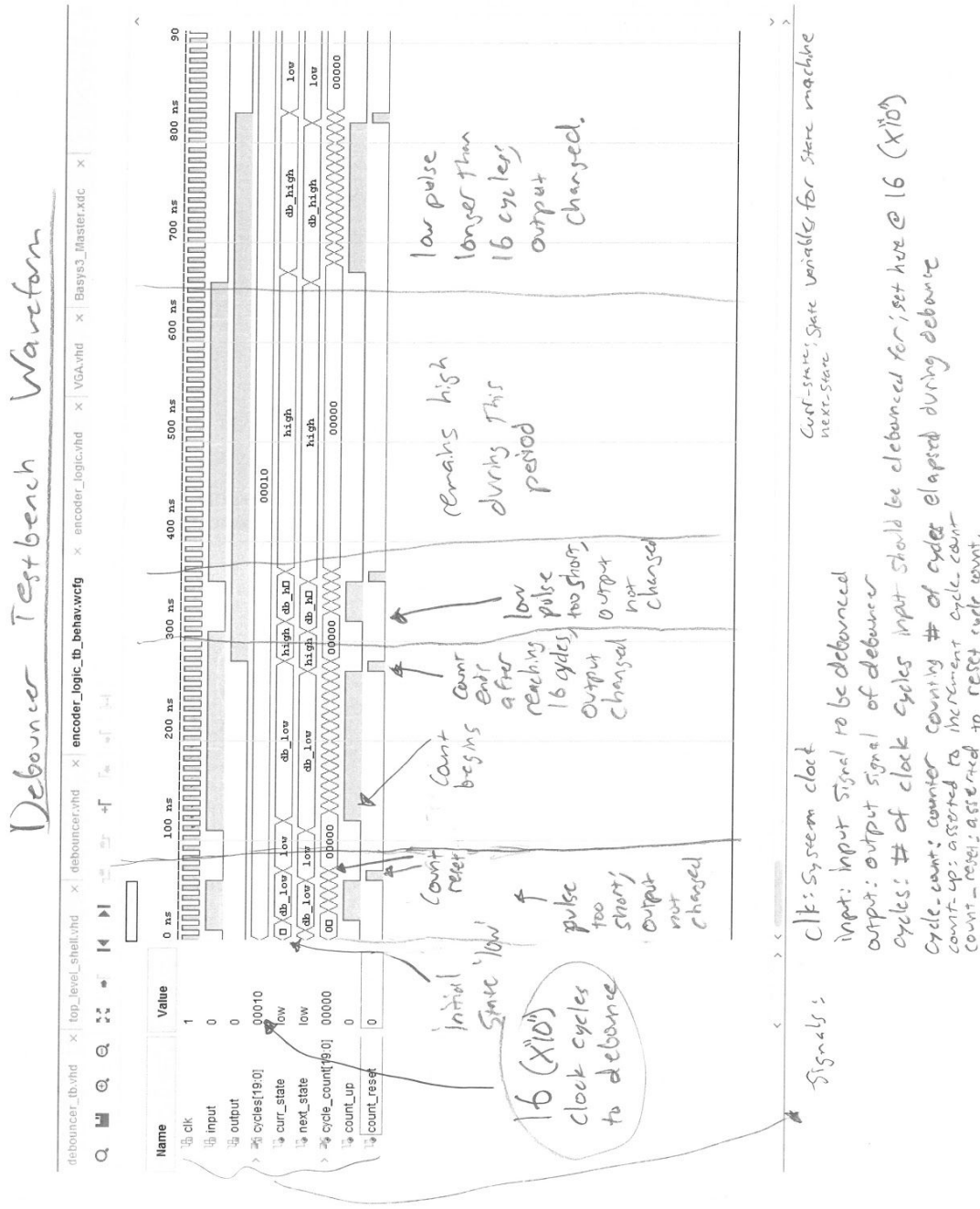
Appendix H: Waveform Graphs

Figure 3: Encoder: Demonstrates basic functionality of Encoder module indicating detection of CW and CCW rotation, with edge case testing towards end.



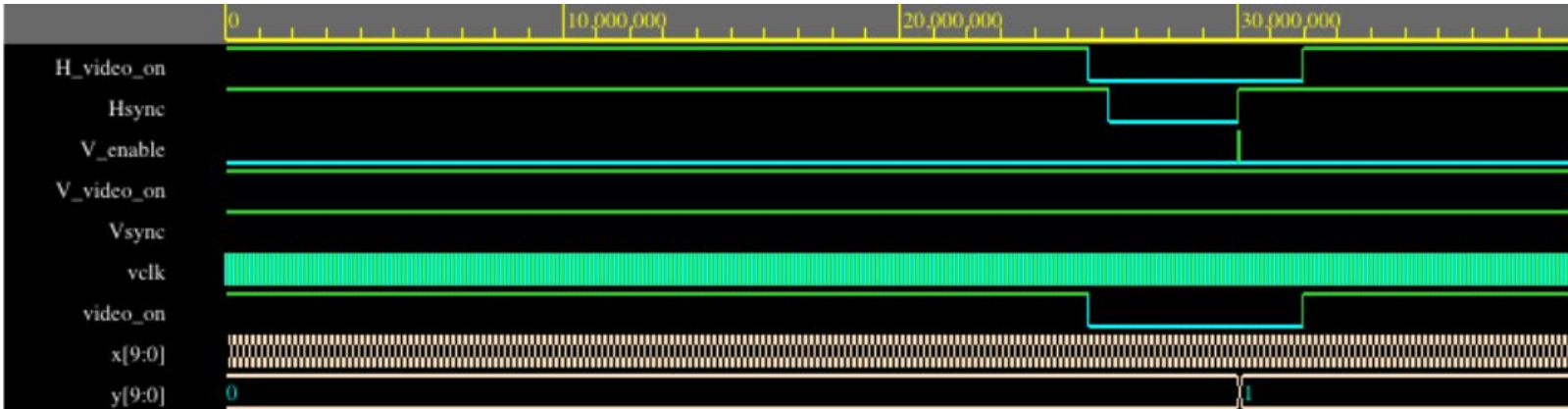
Appendix H: Waveform Graphs

Figure 4: Debouncer: Waveform demonstrating testing of debouncer, showing debounced output only asserted after input asserted for desired number of clock cycles, equal to input 'cycles'



Appendix H: Waveform Graphs

Figure 5: VGA: These waveforms were taken from the solution to the Day 23 in-class exercise and include all relevant testing. The top illustrates the horizontal sweep test, while the bottom illustrates the vertical sweep test.



VERTICAL SWEEP SIMULATION

