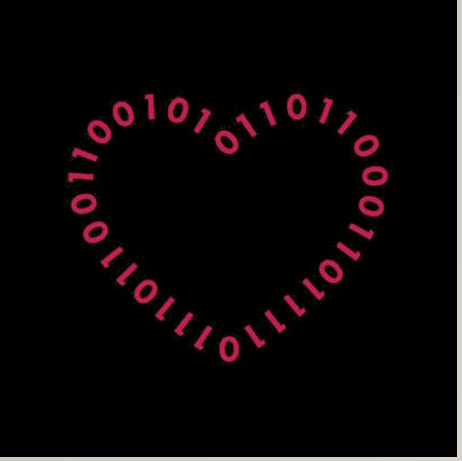
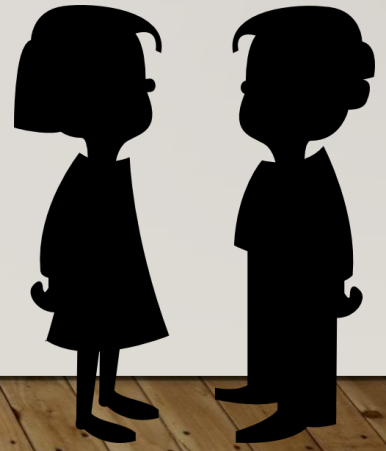


# BINARY LOVE

## THE STORY OF TWO LONELY STICK FIGURES IN A FPGA WORLD

CARLA GERST, XIANGYUAN GUO, KELLEY HARRIS, XINYANG JIAN



# INTRODUCTION

The purpose of this project was to create a program that allowed the user to mask specific pixel ranges which will hide certain parts of a single image at different times throughout the story. The story is told by manually adjusting the switches on the Nexys4 board.

# MATLAB

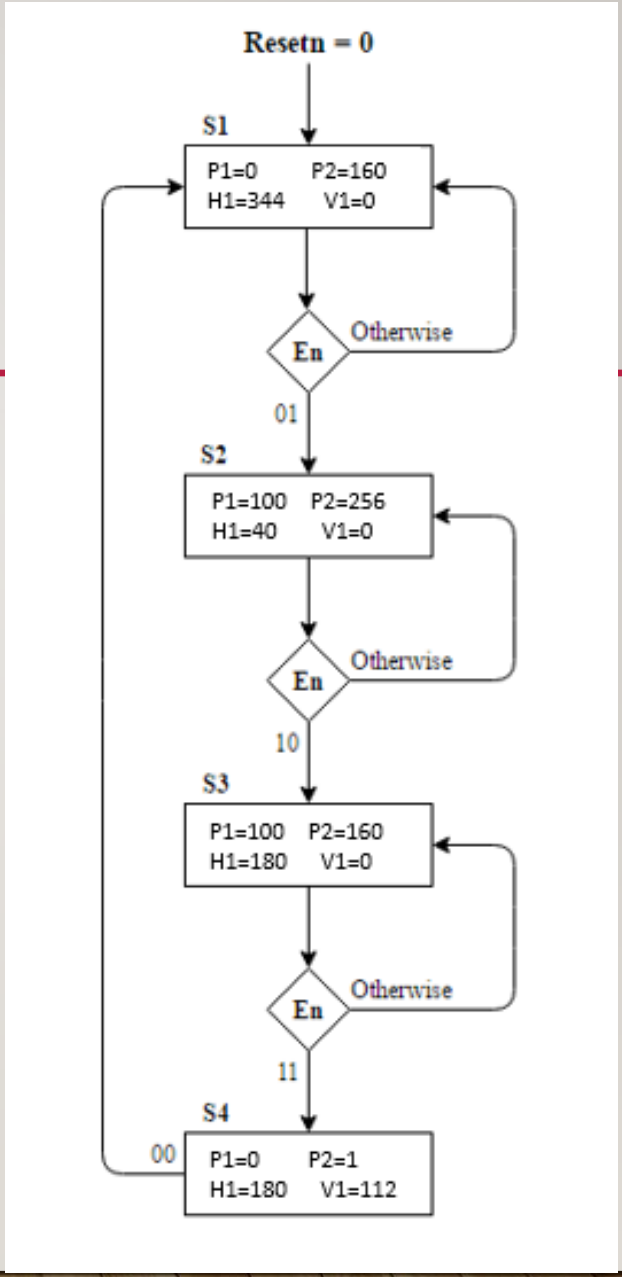
MATLAB was used to convert the PNG image into usable text file.

```
clear all; close all; clc;

I = imread ('binarylove.png'); % RGB image
figure;imshow(I);

% Resizing the image to 256x256:
IP = imresize(I,[256 256]);
figure; imshow (IP);

% 24-bit RGB image: we will convert it to a 12-bit RGB image:
for i = 1:3
    IN(:,:,i) = IP(:,:,i)/16; % every plane converted to 4 bits. right shift
end

figure; imshow(IN*16); % This is just so that 'imshow' can display the image properly

% ------------------------------------------------------------------------
% Converting to text file. Format: 0|R|G|B in hexadecimal
q = quantizer ('ufixed', 'round', 'saturate', [4 0]);
textfile = 'myimg.txt';
fid = fopen (textfile, 'wt'); % generates text file in write mode

for i = 1:256
    for j = 1:256
        R = IN(i,j,1); G = IN(i,j,2); B = IN(i,j,3);
        Rh = num2hex(q, double(R)); Gh = num2hex(q, double(G)); Bh = num2hex(q, double(B));
        fprintf(fid, '0%s%s%s\n',Rh, Gh, Bh);
    end
end

fclose (fid);
```

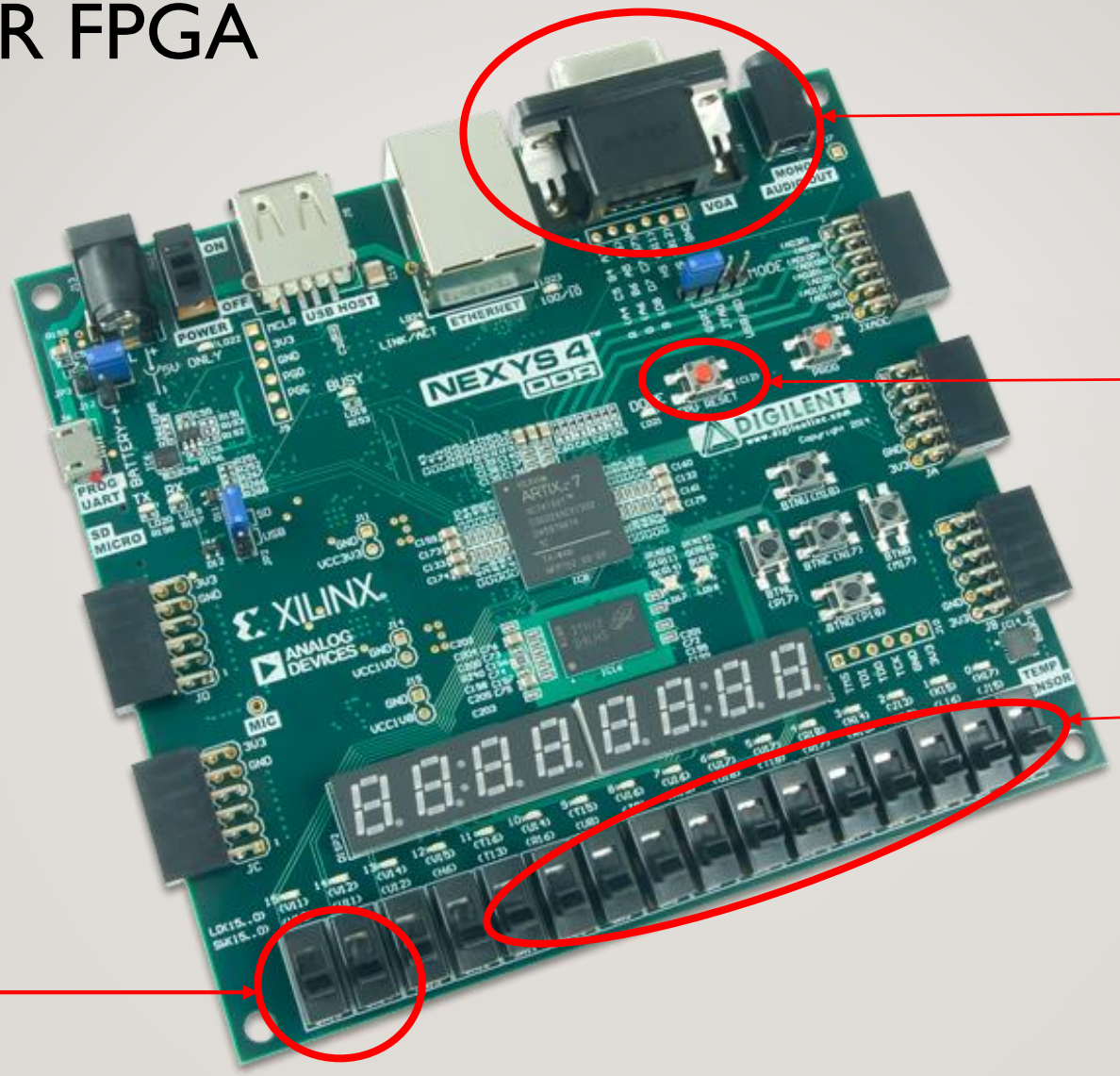
# VGA CONTROLLER & FSM

Once the text file was created, it was then imported onto the VGA controller of the FPGA board. This was done using the provided VGA control program located on the class website.

A Finite State Machine (FSM) was created using the Xilinx program. The state diagram is shown on the right.



# NEXYS 4 DDR FPGA



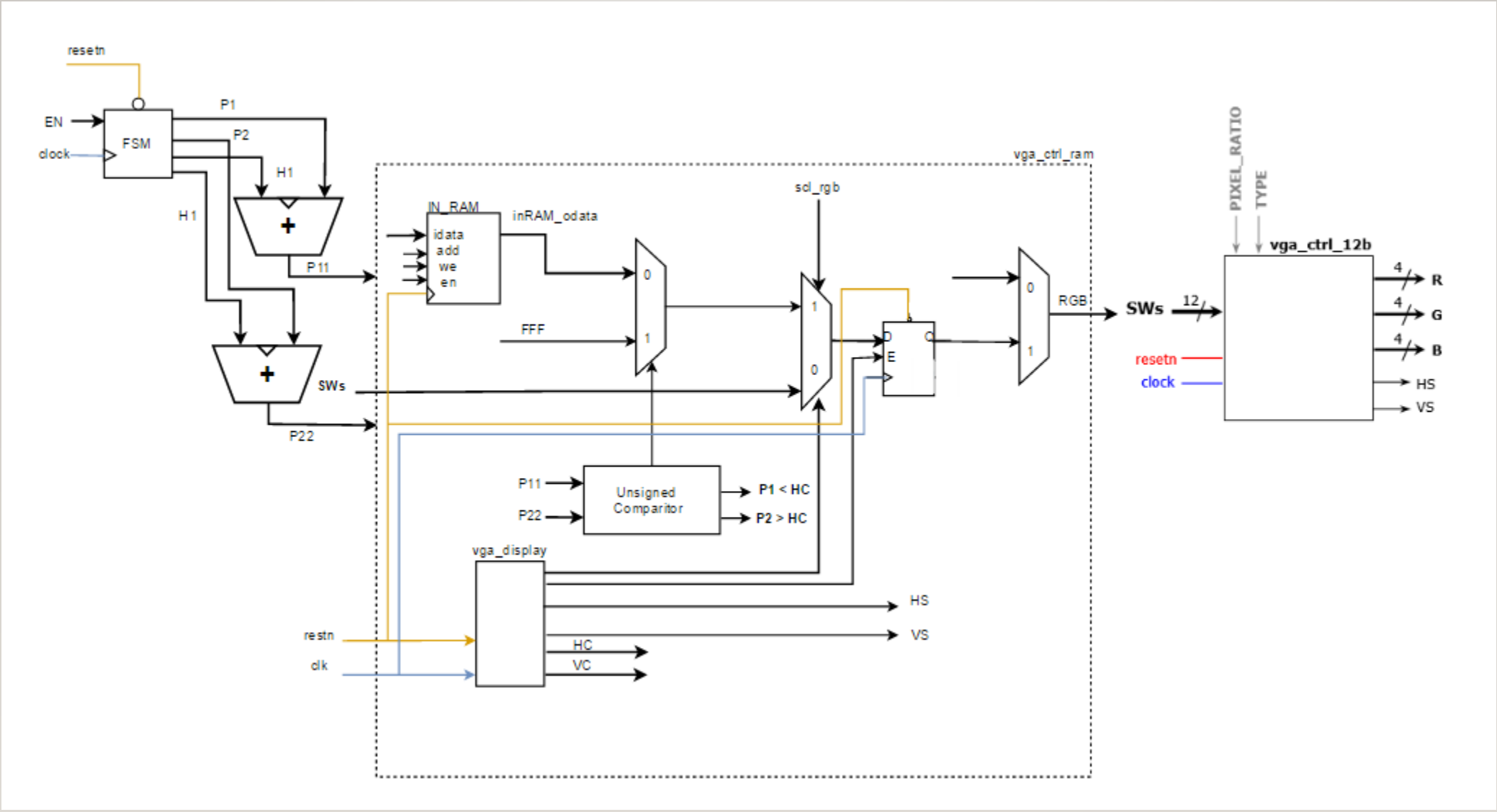

VGA Control

Reset Button

Background
Color Switches

State Switches

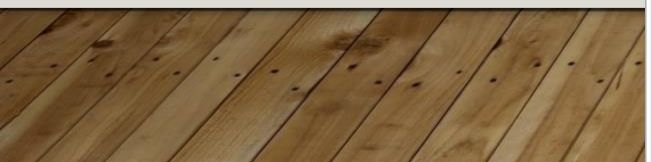
# Datapath Circuit



## FSM CODE

```
architecture Behavioral of FSM_cover is

   type state is (S1,S2,S3,S4);
   signal y: state;


begin

   Transitions: process (resetn, clock, en)
   begin
      if resetn = '0' then -- asynchronous signal
         y <= S1; -- if resetn asserted, go to initial state: S1
      elsif (clock'event and clock = '1') then
         case y is
            when S1 =>
               if en = "01" then y<=S2; else y<=S1; end if;
            when S2 =>
               if en = "10" then y<=S3; else y<=S2; end if;
            when S3 =>
               if en = "11" then y<=S4; else y<=S3; end if;
            when S4 =>
               if en = "00" then y<=S1; else y<=S4; end if;
         end case;
      end if;
   end process;

   Outputs: process (y)
   begin

      case y is
         when S1 -> p1 <= "0000000000"; p2 <= "0010100000";  h1 <= "0101011000"; v1 <= "0000000000"; -- 0/160
         when S2 -> p1 <= "0001100100"; p2 <= "0100000000";  h1 <= "0000101000"; v1 <= "0000000000"; --100/256
         when S3 -> p1 <= "0001100100"; p2 <= "0010100000";  h1 <= "0010110100"; v1 <= "0000000000"; --100/160
         when S4 -> p1 <= "0000000000"; p2 <= "0000000001";  h1 <= "0010110100"; v1 <= "0001110000"; --none
      end case;
   end process;

```

## CHANGES TO PROVIDED VGA CODE



```
iram: in_RAM generic map (NDATA => NDATA, FILE_IMG => FILE_IMG)
      port map (clock, resetn, inRAM_idata, inRAM_add, inRAM_we, inRAM_en, inRAM_odata);

      -- Writing to this RAM is disabled here, but we can always enable to load new data
      inRAM_idata <= (others => '0'); inRAM_en <= '1'; inRAM_we <= '0';

      -- The following formulas only works if the image is square and the width size is a power of 2.


      inRAM_int <= (conv_integer(vcount_buf)-conv_integer(v1))*256+ (conv_integer(hcount_buf)-conv_integer(h1));
      inRAM_add <=conv_std_logic_vector(inRAM_int, 16);

      -- This is how we control what appears on the screen (only a square 2**NBPR x 2**NBPR)
      sel_RGB <= '1' when (h1 < hcount_buf and hcount_buf < 2**NBPR+h1) and (v1< vcount_buf and vcount_buf < 2**NBPR+v1) else '0';

      in_RGB <= X"FFF" when (p1 < hcount_buf and hcount_buf < p2) else inRAM_odata (11 downto 0);

      with sel_RGB select
         |   in_RGB2 <= in_RGB when '1', -- only the 12 LSBs contain useful data
                       sw when others;
```

# Enjoy the STORY!

Boy: It's lonely being a single bit in this FPGA world. No one to walk through AND gates, OR gates. I've never even seen an XOR gate. I wonder if I'll ever find another bit to move through this integrated circuit.

Girl: I walk around wondering if I'll ever meet another bit. Even moving in an address bus to a multiplexer, there's no one to count on.

Narrator: One day, the girl bit gets on the address bus, and sees that she's not alone. She decides to use the shift register to move next to the boy.

Girl: "Hey, boy, I like your gray code".
Boy: "Sequence detected. Would you like to go through the XOR gate with me?

Narrator: Binary Love is like an asynchronous circuit.  Eventually, your circuit will get implemented. AWE!