Nexys Racer

ECE 4710

Winter 2022

Alexander Saikalis

Shane MacFadyen

David Stamatovski

Seeyam Chowdhury

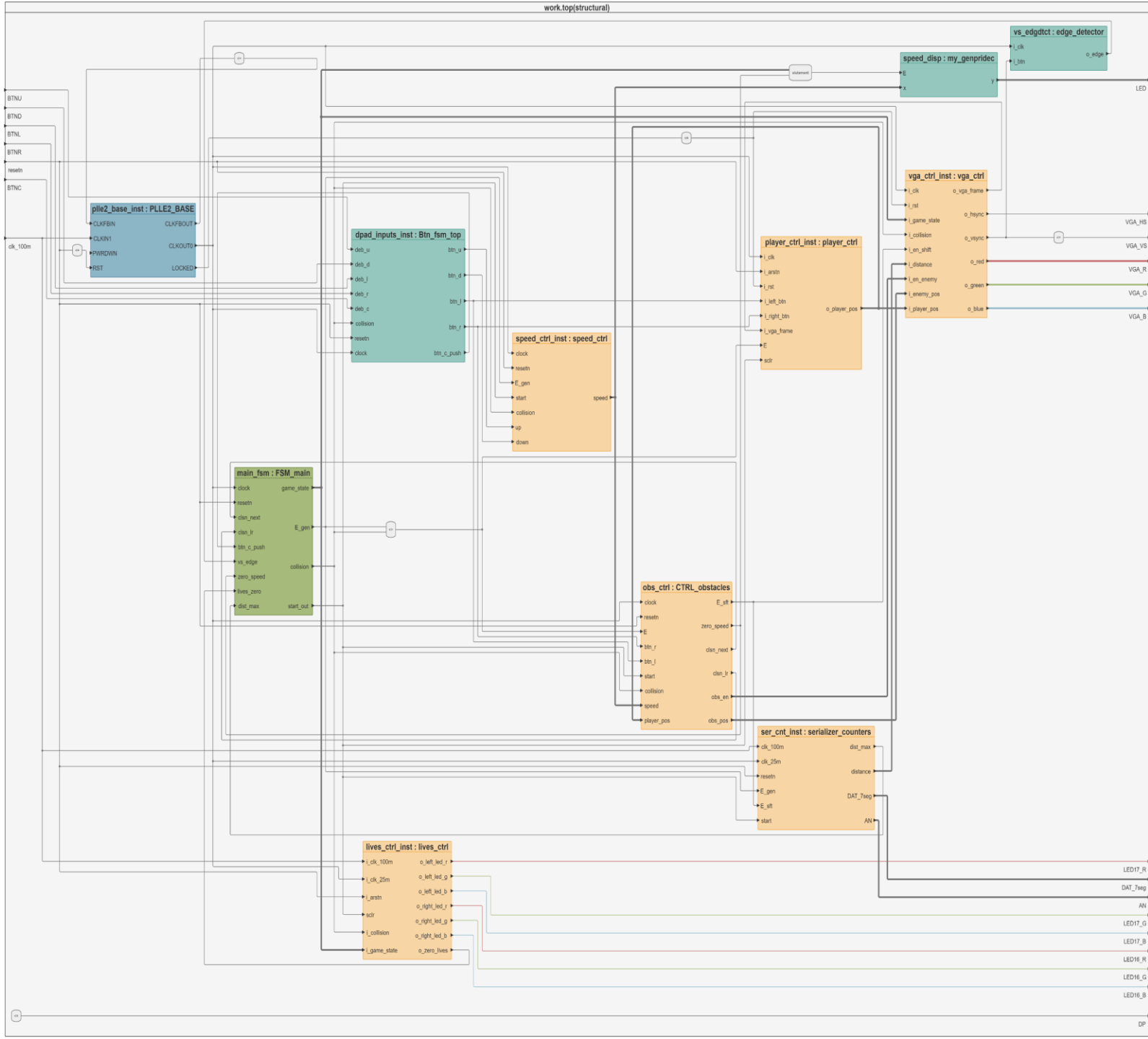
Project Description

User controls a car object using the FPGA's D-pad in a multiple lane road. The user's point of view will be from the top. Other objects, such as cars, will be in other lanes to serve as obstacles. The screen and objects will scroll relativistically based on the speed of the user's car.

- General Gameplay
  - Avoid obstacles
  - Reach end in fastest time possible
  - Game ends when reaching finish or the player loses all their health

- Peripherals
  - Up/Down Buttons: Accelerate/Decelerate
  - Left/Right Buttons: Move Player Left/Right
  - Center Button: Navigate Menus/Pause
  - RGB LEDs: Lives
  - Standard LEDs: Speed Bar
  - 7-Seg Display: Distance and Time
  - VGA: Video Graphics

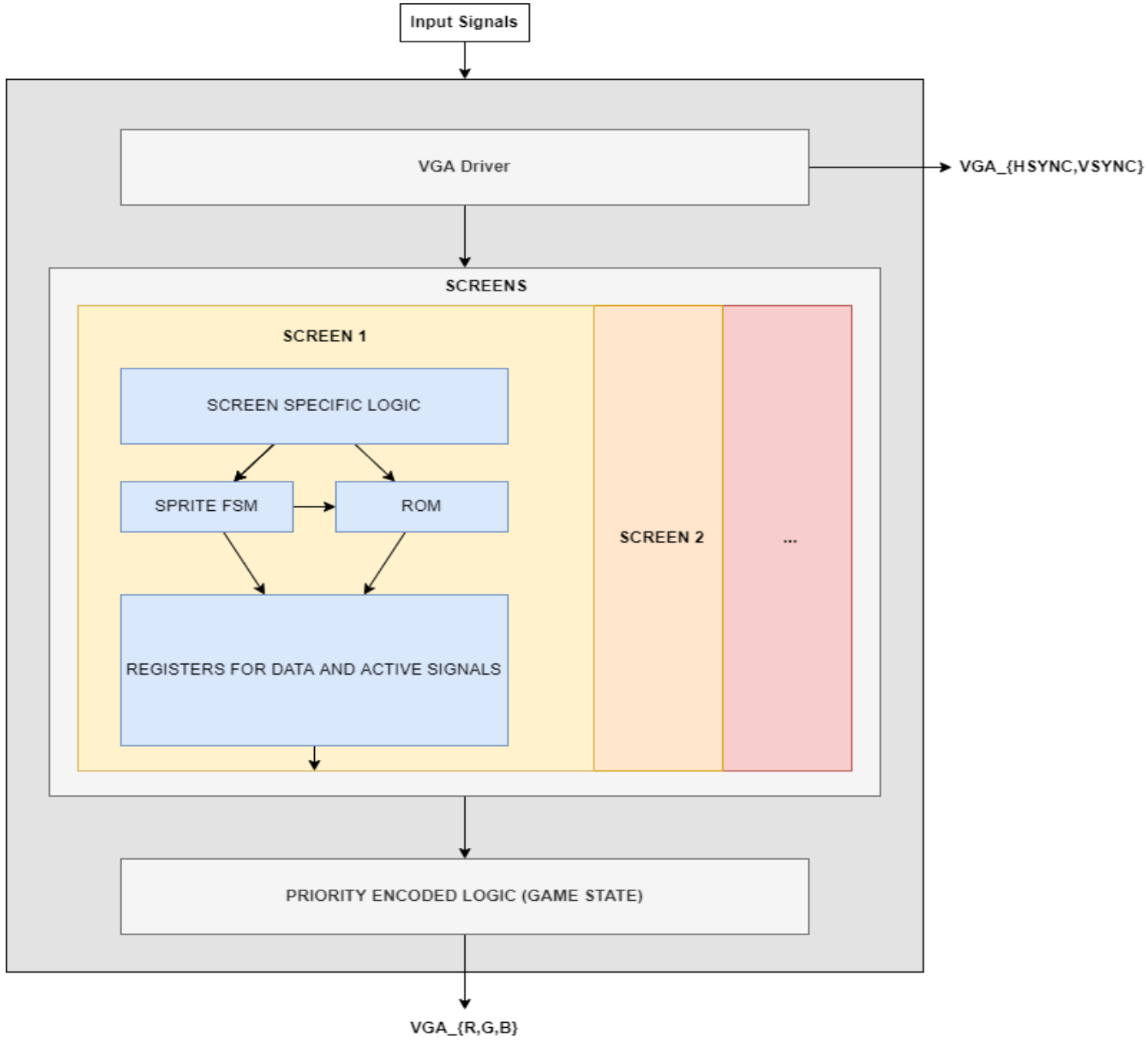
Top Block Diagram



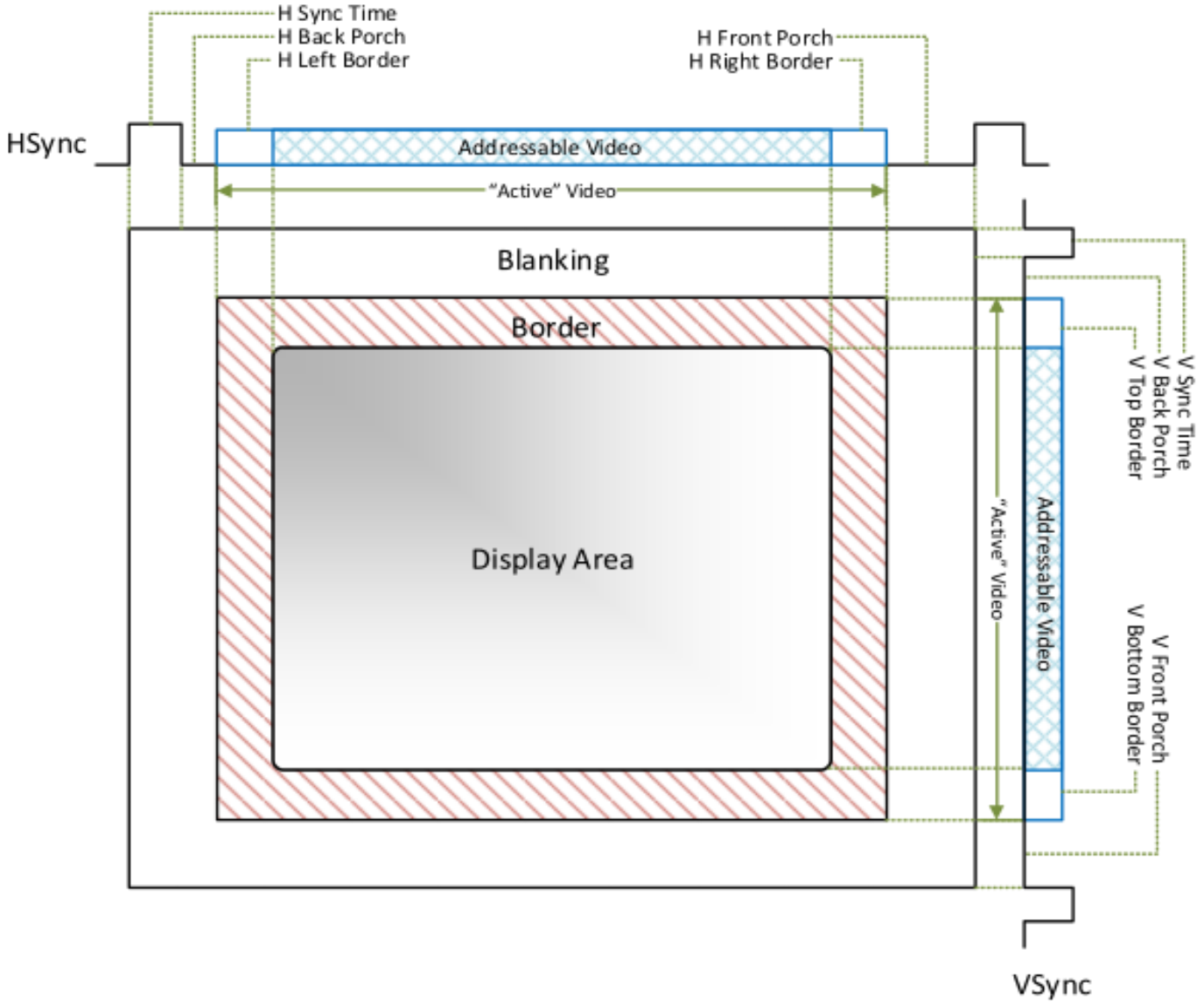
Top Block Diagram Components

- Datapath
  - IO Management
    - Button Conditioning – Btn_fsm_top
    - Speed Display – Priority Decoder
    - Distance and Time 7-Seg Serializer
    - PLL Clock Manager – 25 MHz Clock
  - User Controlled Components
    - Speed/Player Control
  - Game Controlled Components
    - Obstacle Control
    - Lives Control
  - Screen Output Components
    - VGA Controller
- Main Control Circuit
  - Main FSM
  - Edge Detector – VS output VGA signal

VGA Controller

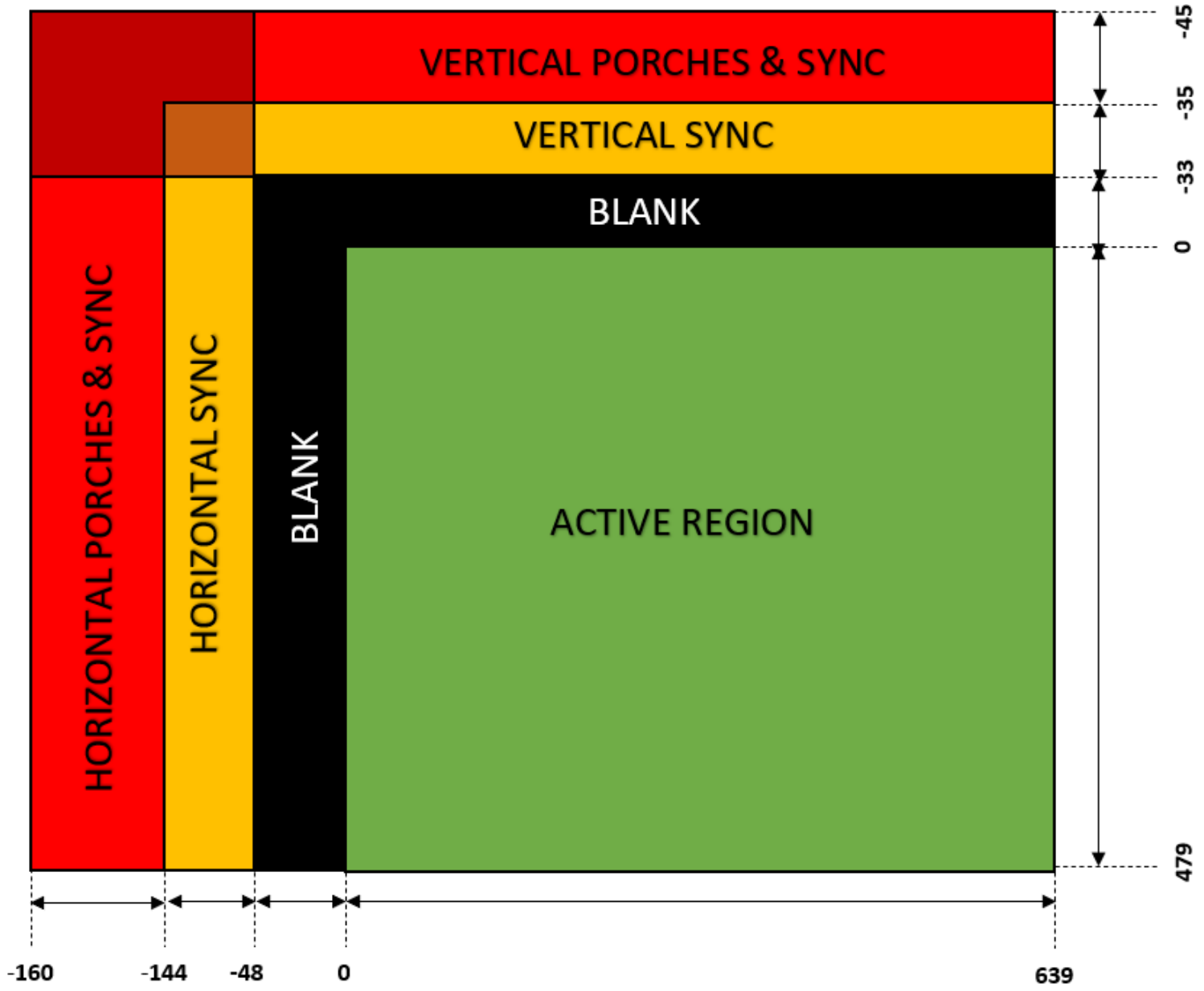


VGA Driver (Method #1)



Source: https://digilent.com/reference/learn/programmable-logic/tutorials/vga-display-congroller/start

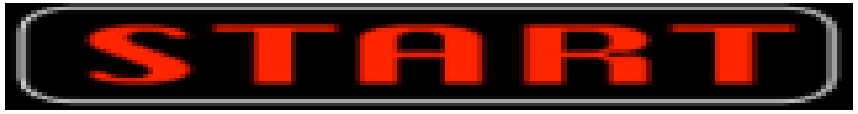
VGA Driver (Method #2)



Sprite Generation

- Sprites are stored in ROM
- Based on the structure of a VHDL file, Vivado can infer that it should be placed in memory
- A Python script was created to:
  - Extract all RGB pixel data from an image for a given resolution
  - Output a formatted VHDL file that can be inferred as ROM
- Data retrieval is only available at each rising edge of the clock

Sprite Generation Example

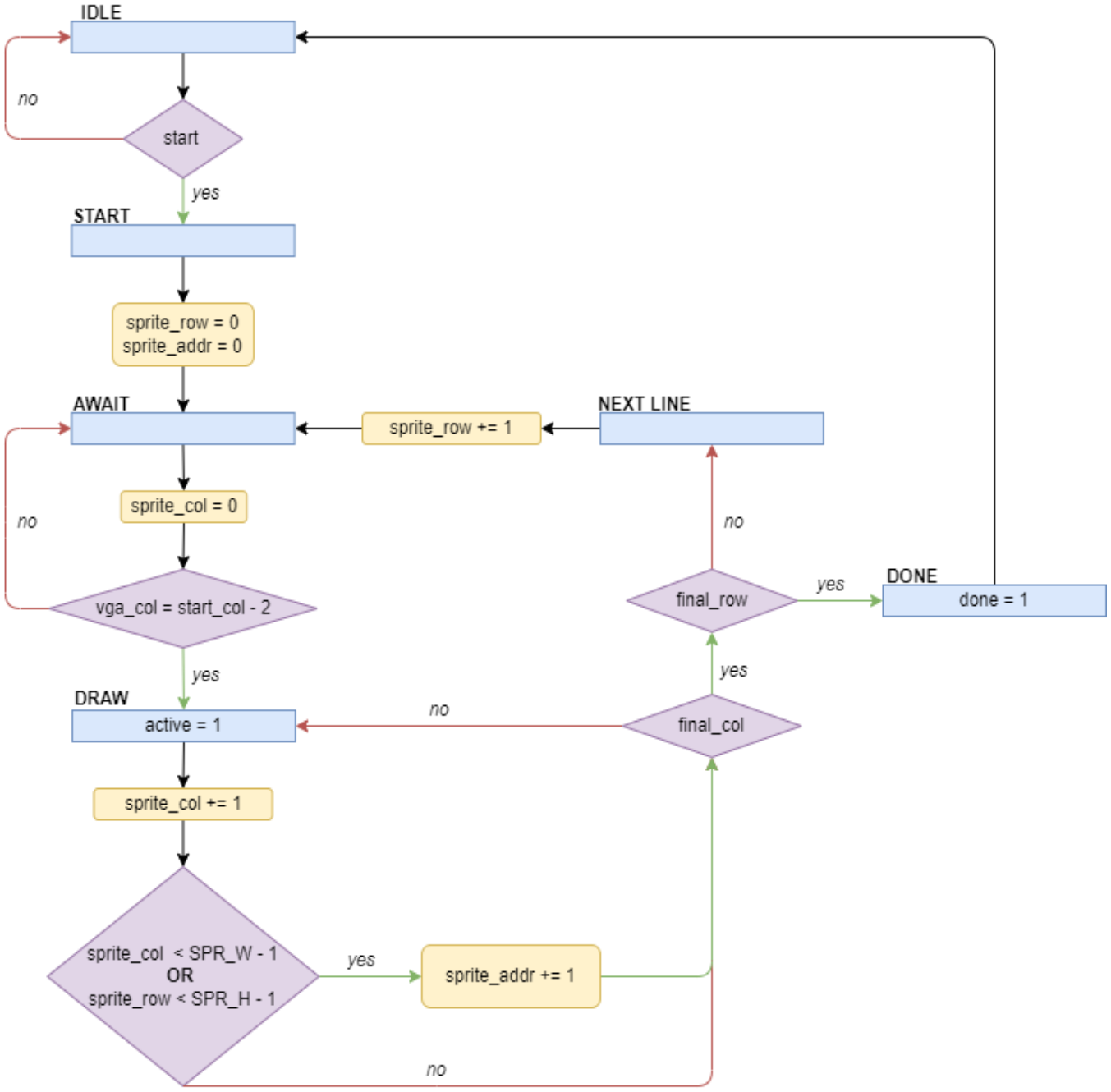



```python
# Read image using RGB encoding
img = iio.imread(file_path, mode="RGB")
img = Image.fromarray(img, mode="RGB")

# Calculate bit width and depth of address bus
col_width = math.ceil(math.log2(img.width))  # bit width per column
row_width = math.ceil(math.log2(img.height))  # bit width per row

bin_repr_vec = np.vectorize(np.binary_repr)

# Create address data with values 0 to (col_width * row_width - 1) and
data = np.arange(0, img.width * img.height)
bin_data = bin_repr_vec(np.array(data), width=(col_width + row_width))

# Convert RGB values to 4-bit binary, concatenate, and create a list of
rgb = []
for x in range(img.height):
    for y in range(img.width):
        pix_data = img.getpixel((y, x))
        r = "{0:04b}".format(pix_data[0])[0:4]
        g = "{0:04b}".format(pix_data[1])[0:4]
        b = "{0:04b}".format(pix_data[2])[0:4]
        rgb.append(r + g + b)

# Create the VHDL BRAM file
f = open(file_name + "_rom.vhd", "w")

f.write("--Generated with bram_generator.py\n\n")
f.write("library ieee;\n")
```

```vhdl
--Generated with bram_generator.py

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

entity red_start_rom is
    port(
        i_clk     : in std_logic;
        i_addr : in std_logic_vector(12 downto 0);
        o_data    : out std_logic_vector(11 downto 0)
    );
end red_start_rom;

architecture behavioral of red_start_rom is

    type rom_type is array (0 to 4799) of std_logic_vector(11 downto 0);
    signal rom : rom_type := (
                    "000000000000",
                    "000000000000",
```

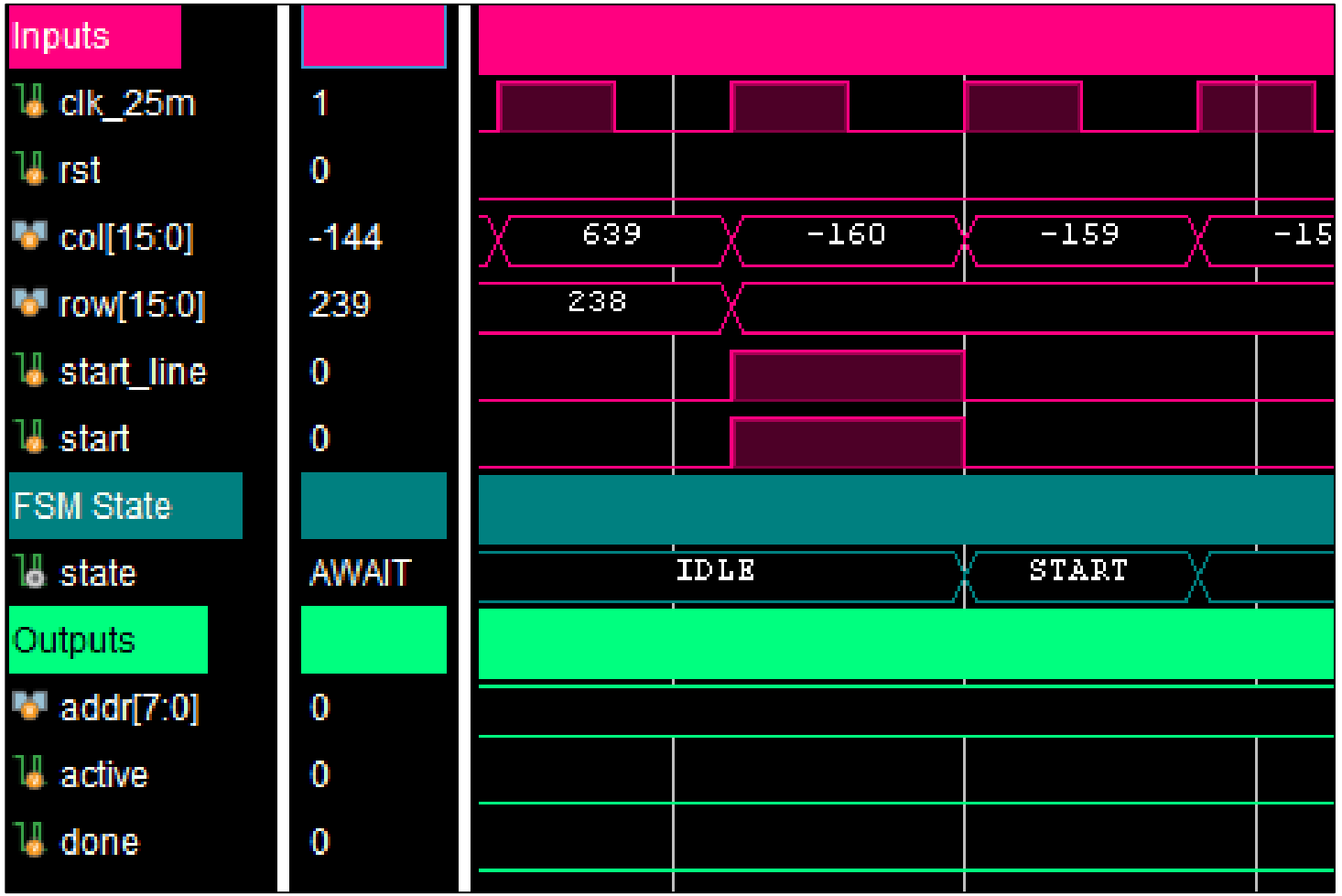
Sprite FSM

- FSM mirrors VGA column and row iteration
- Once start is triggered, each address of the sprite is output on proceeding rising edges (one at a time)
- Sprite address, column, and row are registered values
- Active signal only when the screen is drawing
- Conditional statements are used to ensure correct increase in position

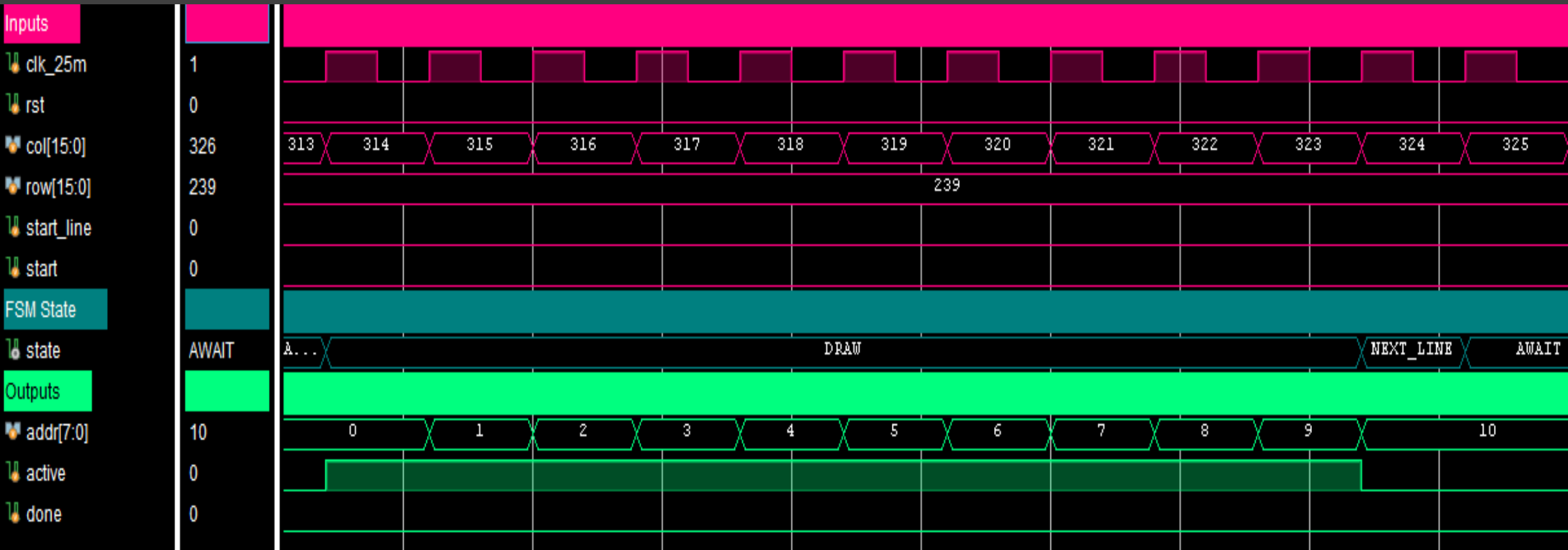


Sprite FSM Example (START)

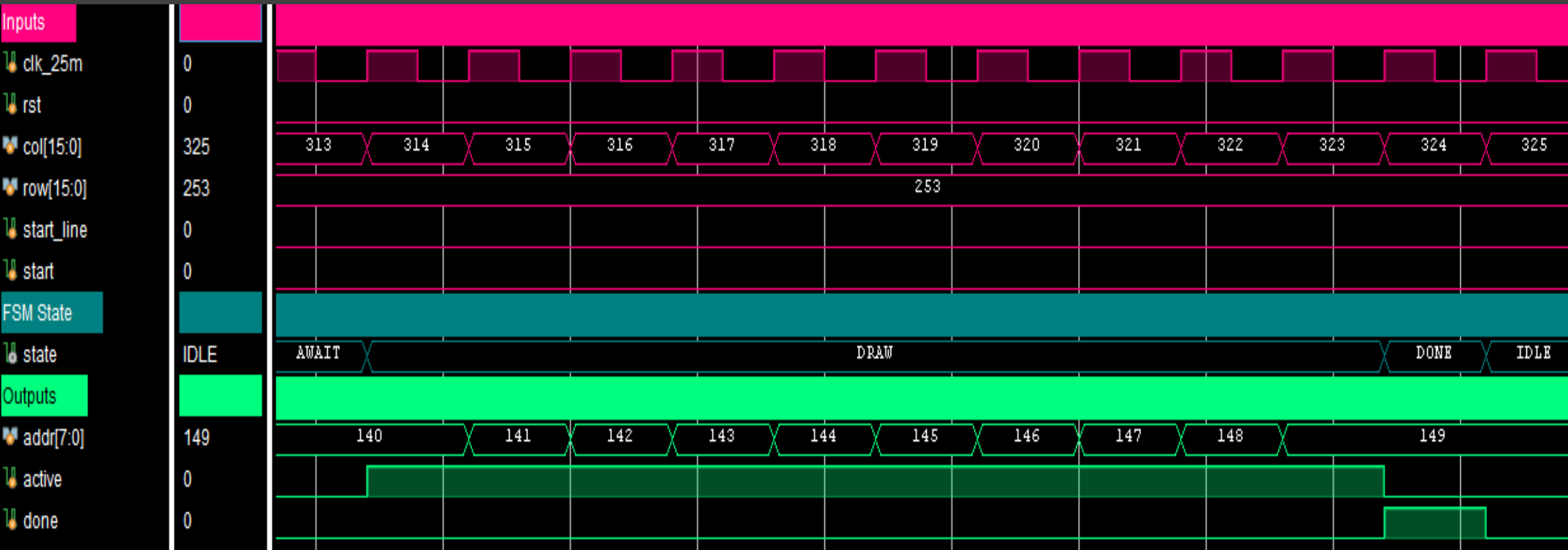
- Assume a sprite with the following dimensions:
  - Width: 10 px
  - Height: 15 px
- To center the sprite:
  - Start when start_line = ‘1’ and VGA row position = 239
  - Start column: $\frac{639-10}{2} = 314$



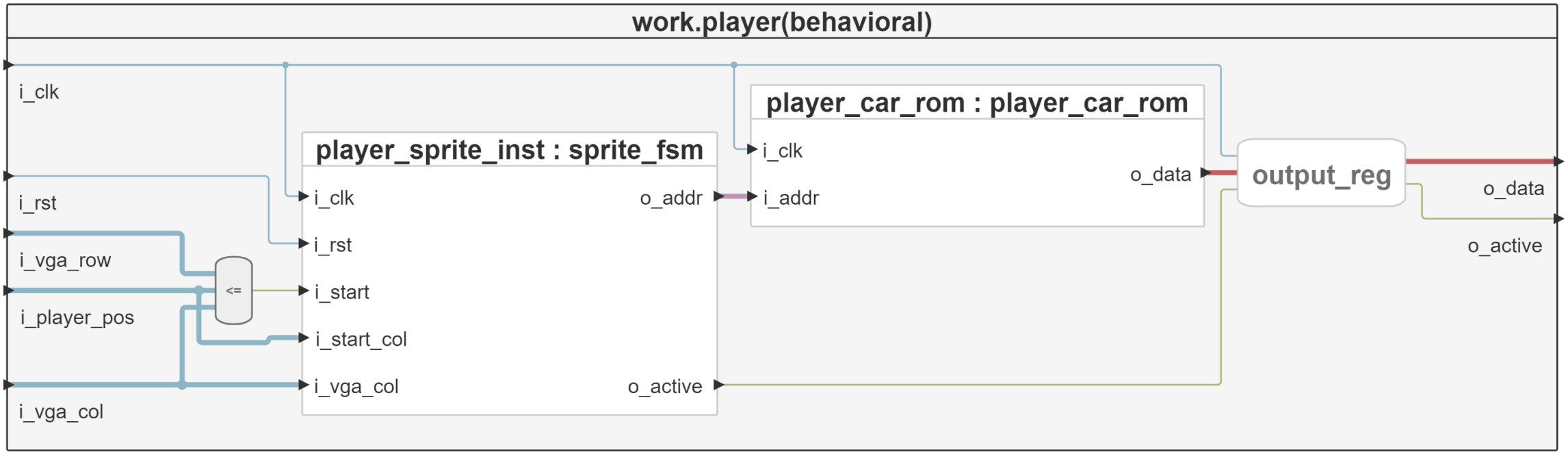
Sprite FSM Example (DRAW)



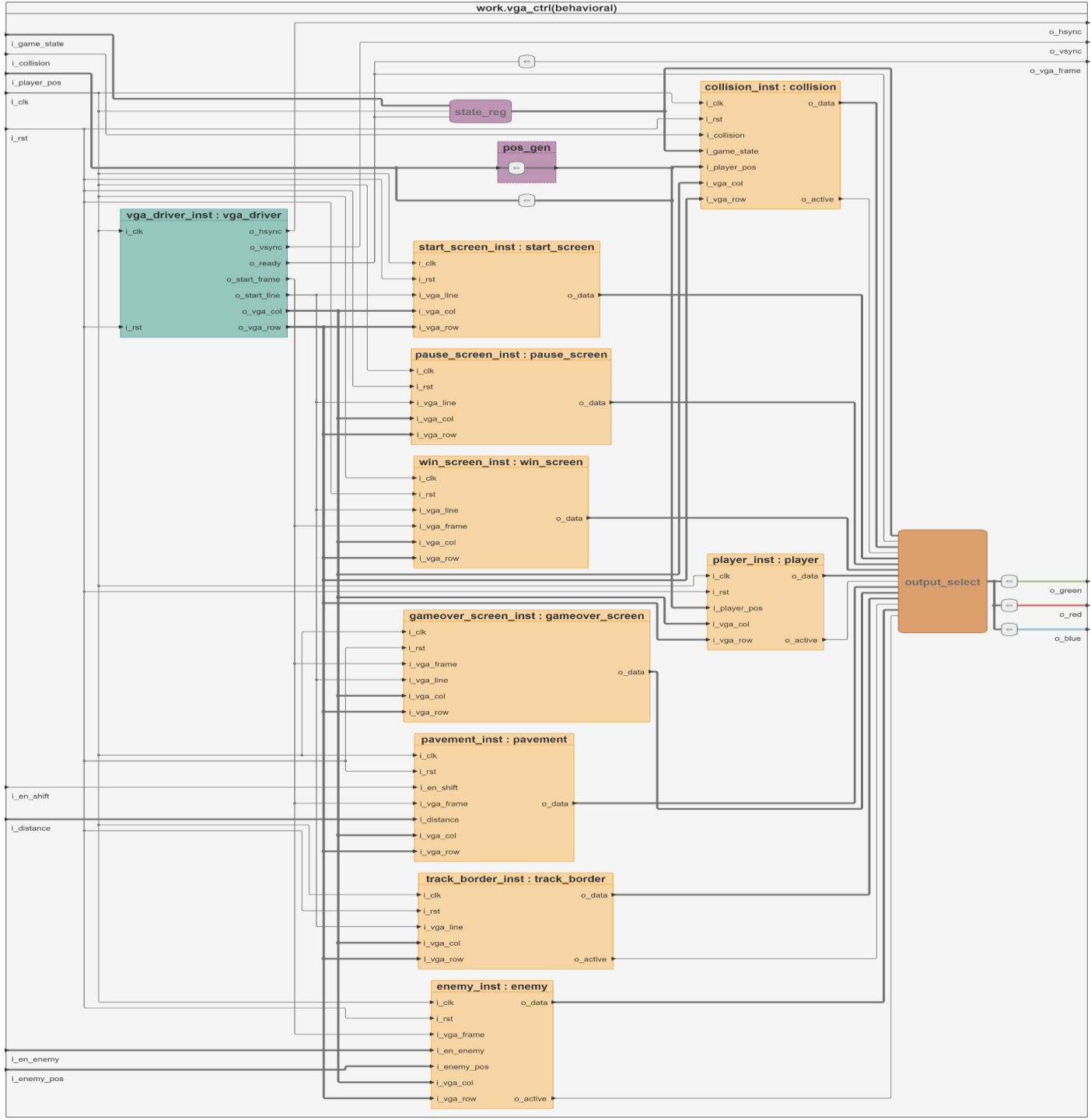
Sprite FSM Example (DONE)



Example Screen Structure



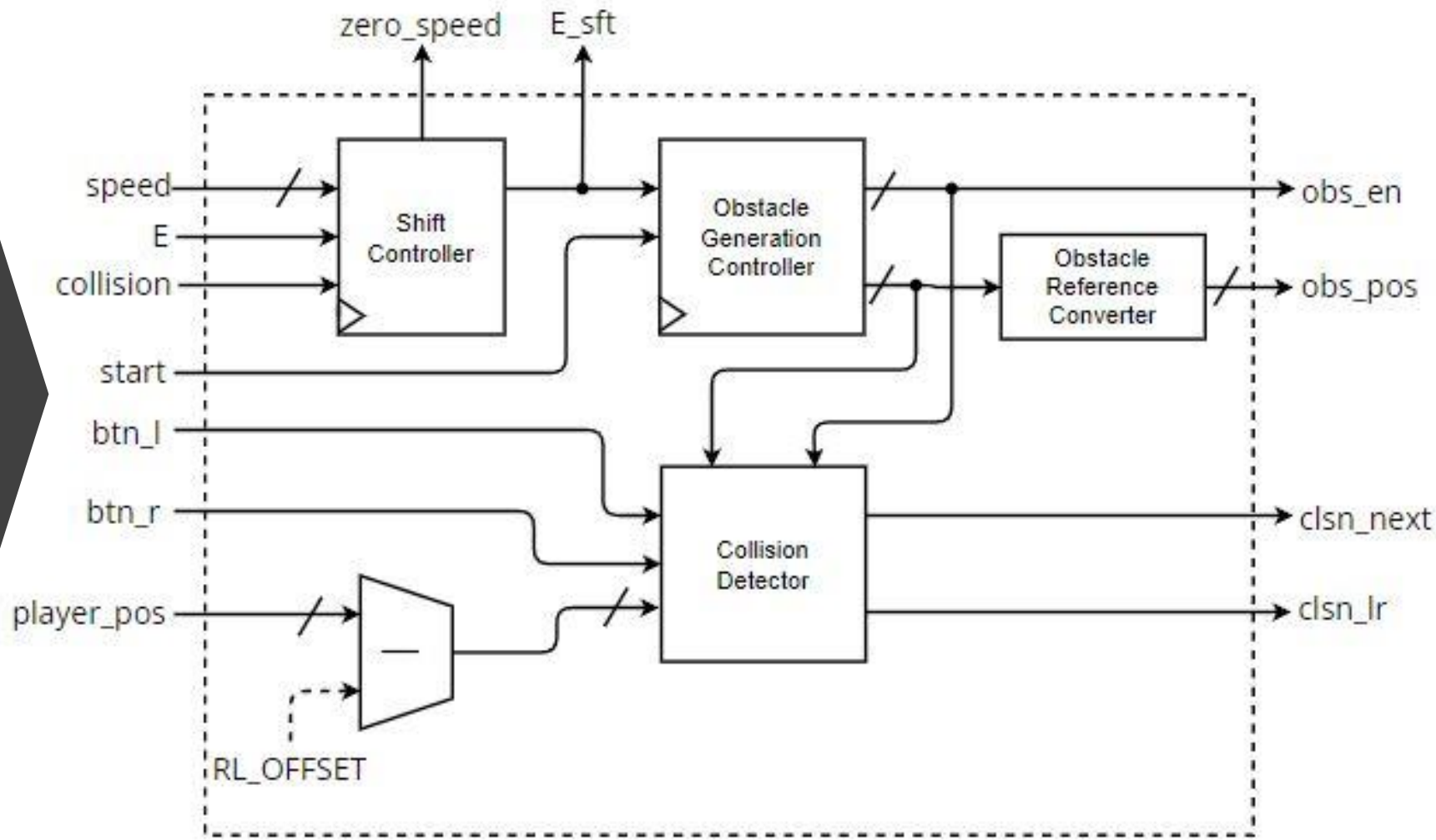
VGA Controller Block Diagram



Obstacle Control

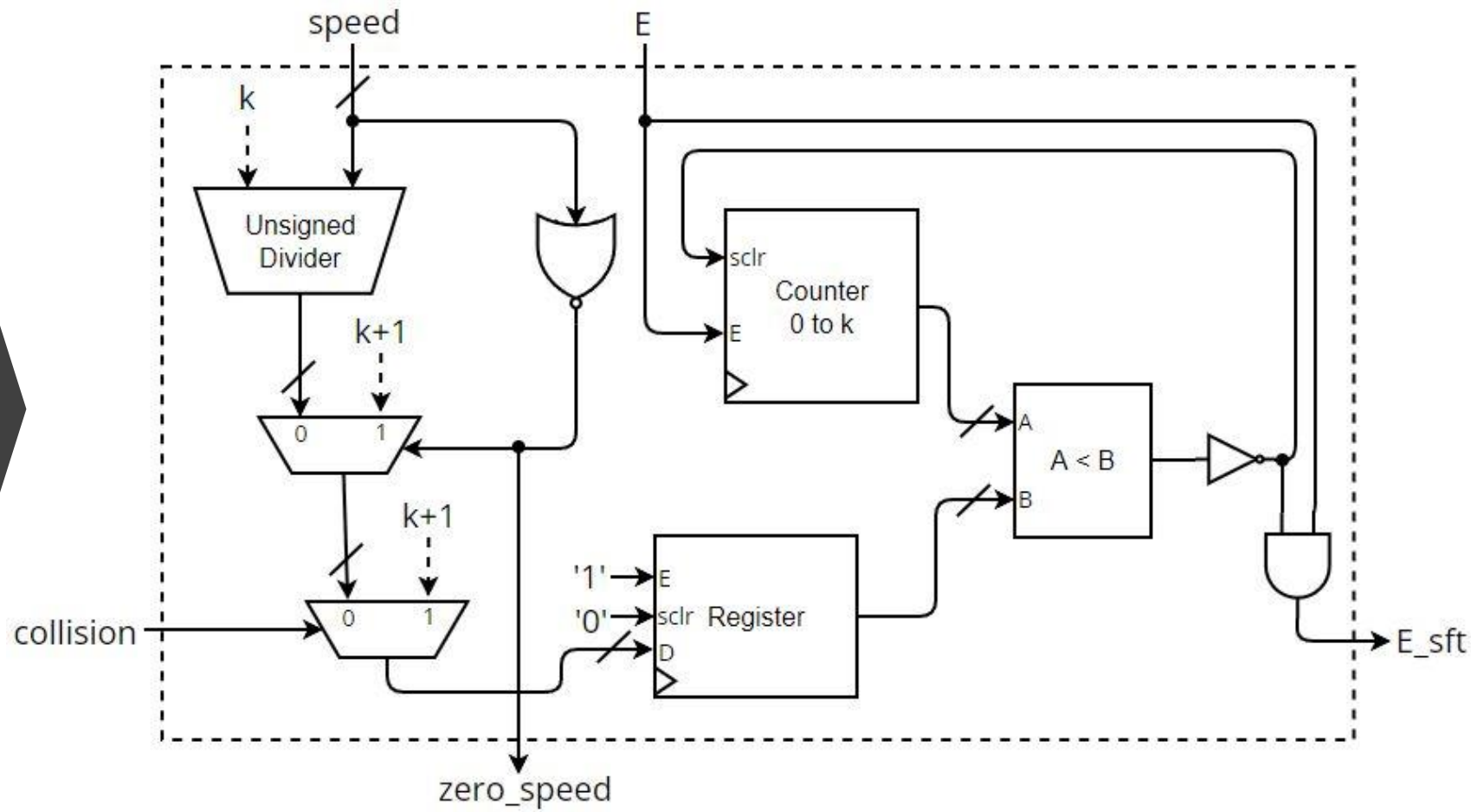
- Controls the generation of obstacles and all their related signals
  - Creates a shift enable signal (E_sft) that instructs other blocks as to which cycles the obstacles shift
  - Generates a zero-speed signal for when the obstacles are not moving
  - Generates obstacle position signals and enable signals for each lane
  - Determines if a collision with an obstacle is going to occur in front of the player or to the side of the player

Obstacle Control



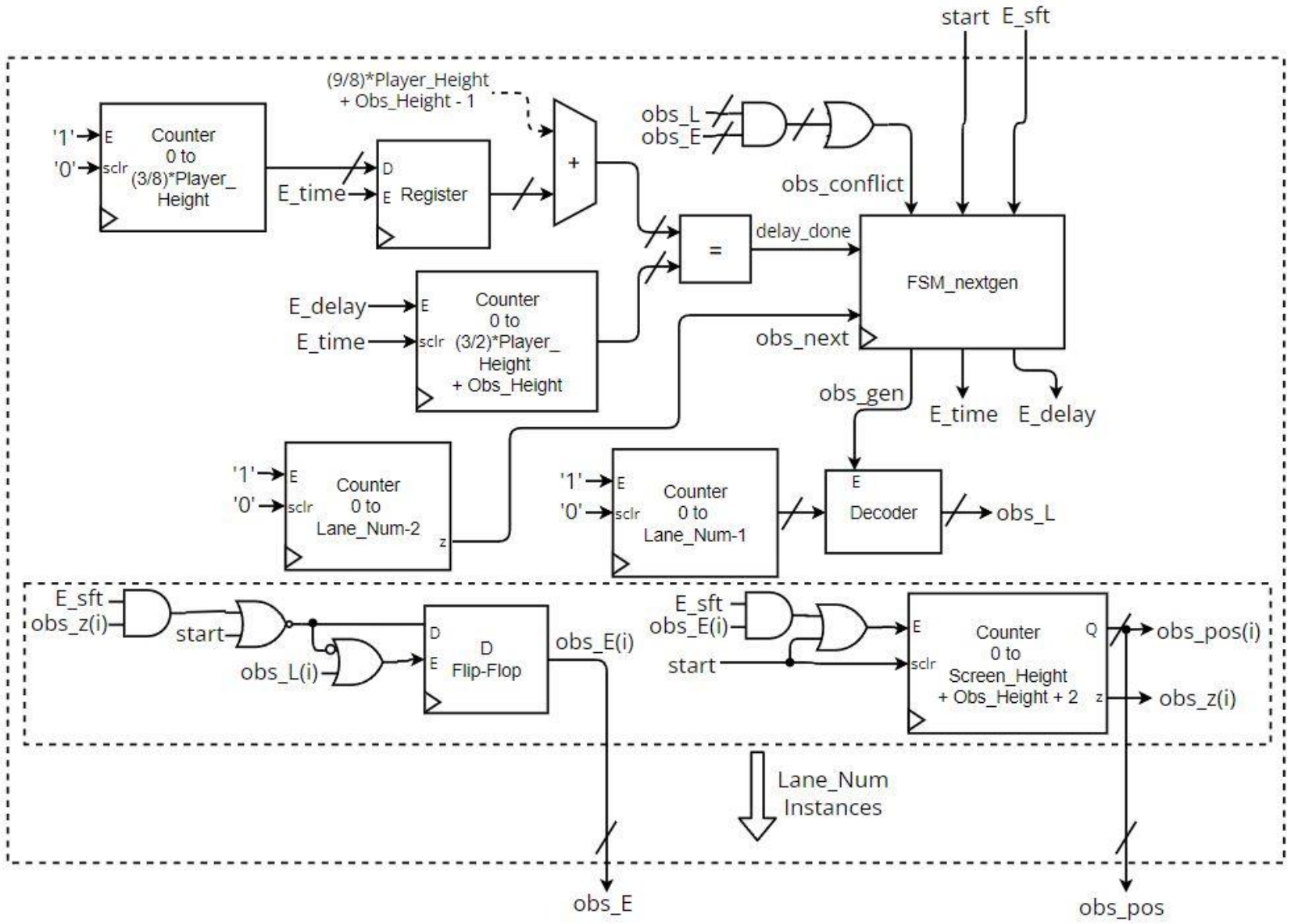
$$\text{RL\_OFFSET} = \frac{\text{DISPLAY\_WIDTH} - \text{LANE\_NUM} \times \text{LANE\_WIDTH}}{2} = 60$$

Shift Control

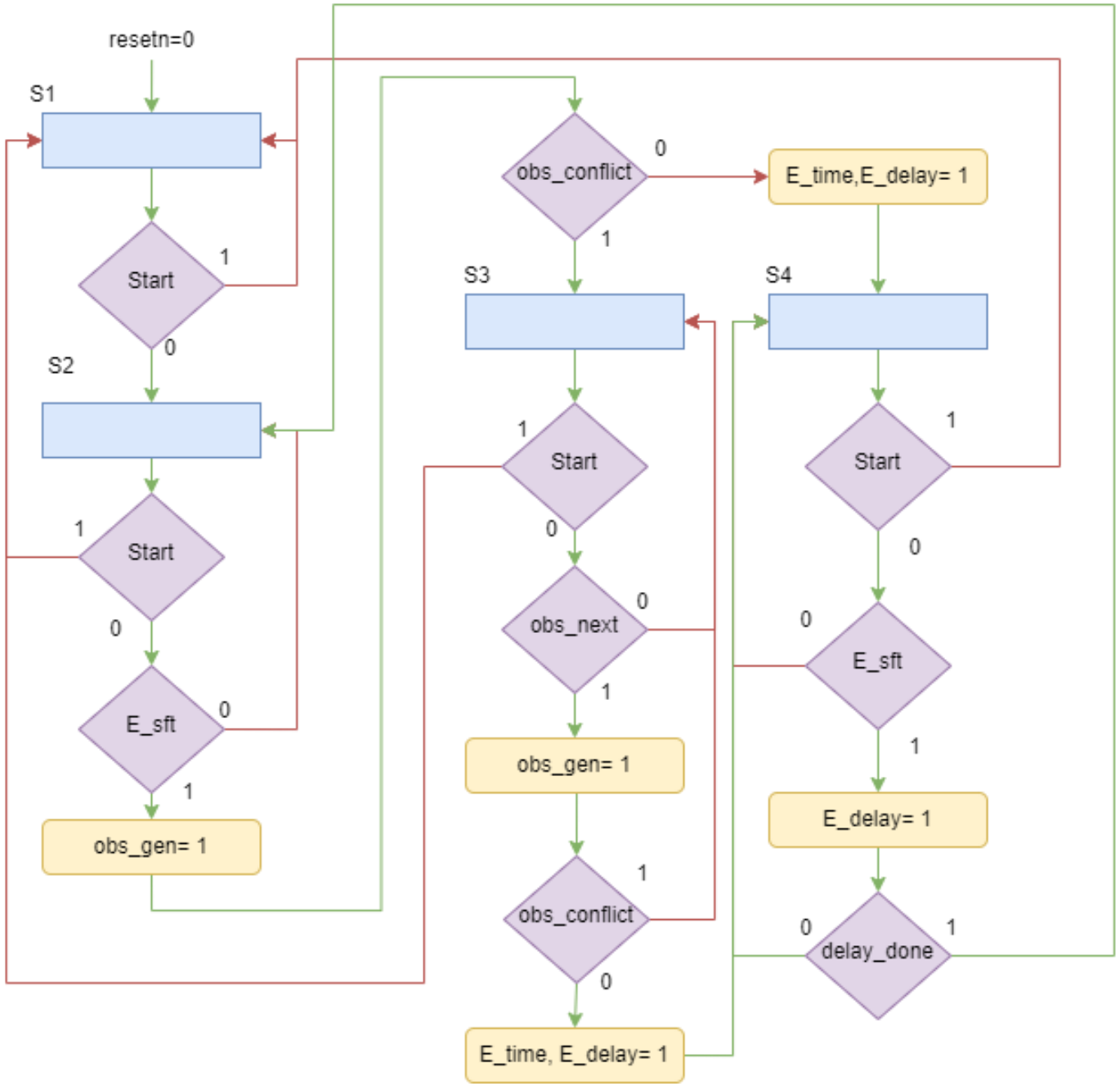


$$k = \frac{\text{MAX\_SPEED} \times 10^9}{\text{REFRESH\_RATE} \times \text{MAX\_SHIFTS\_PER\_CYCLE} \times \text{CLK\_PERIOD\_NS}} = 3.281 \times 10^6$$

Obstacle Generation Controller



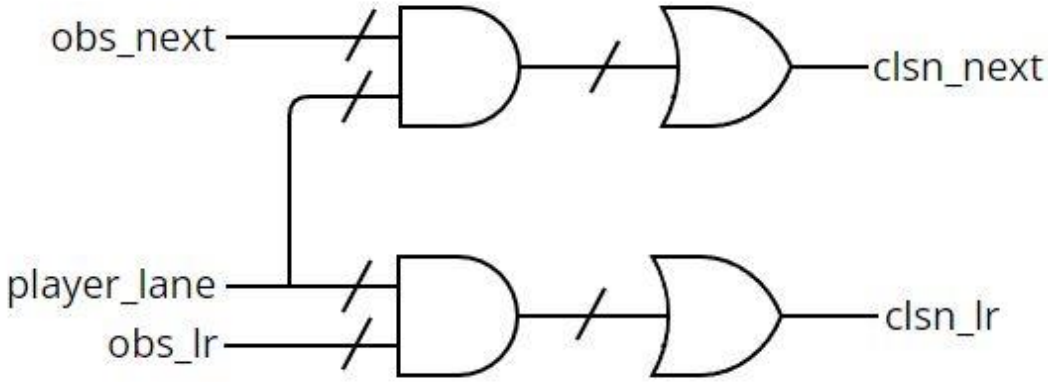
Obstacle Generation FSM



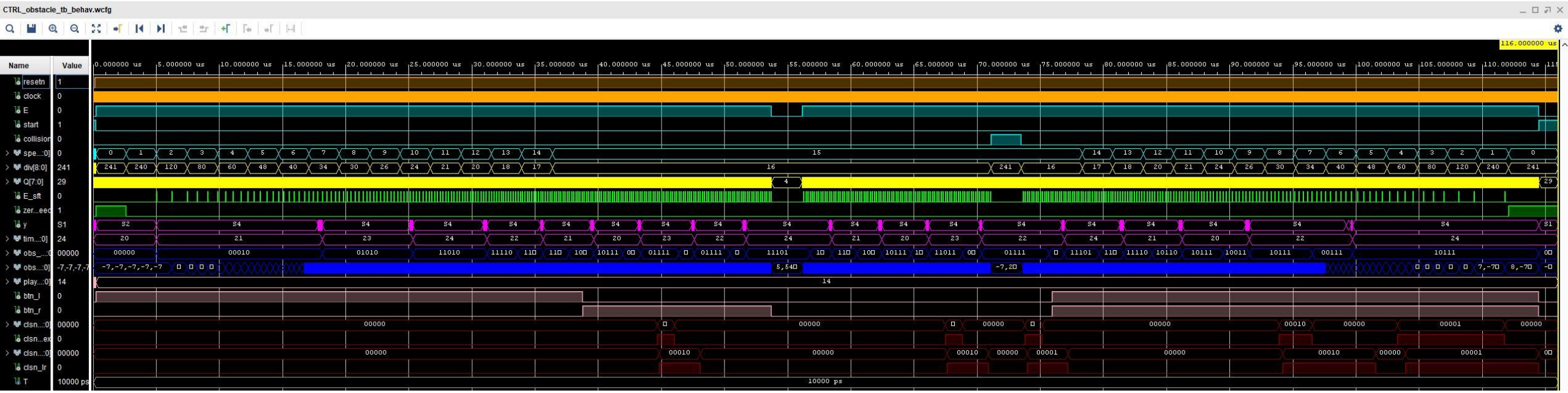
Collision Detector

- Obstacle Process
  - Determines obs_next and obs_lr signals for each lane
  - Obs_next: the obstacle is in front of a potential player position
  - Obs_lr: the obstacle is to the left or right of a potential player position
- Player Process
  - Determines next player position based on the left/right button inputs
  - Determines which lanes the next player position will occupy

- Collision Determination:

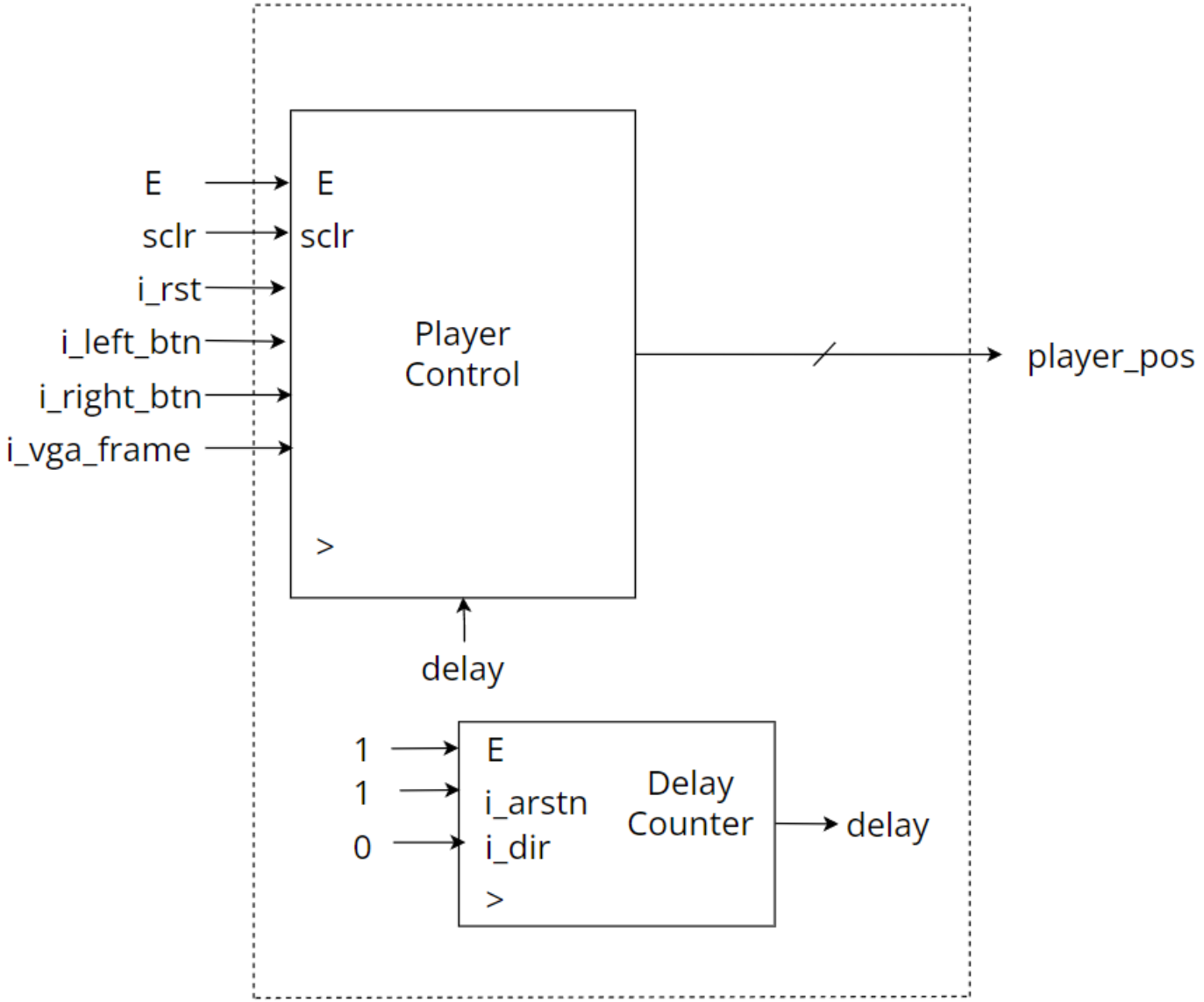


Obstacle Control Simulation



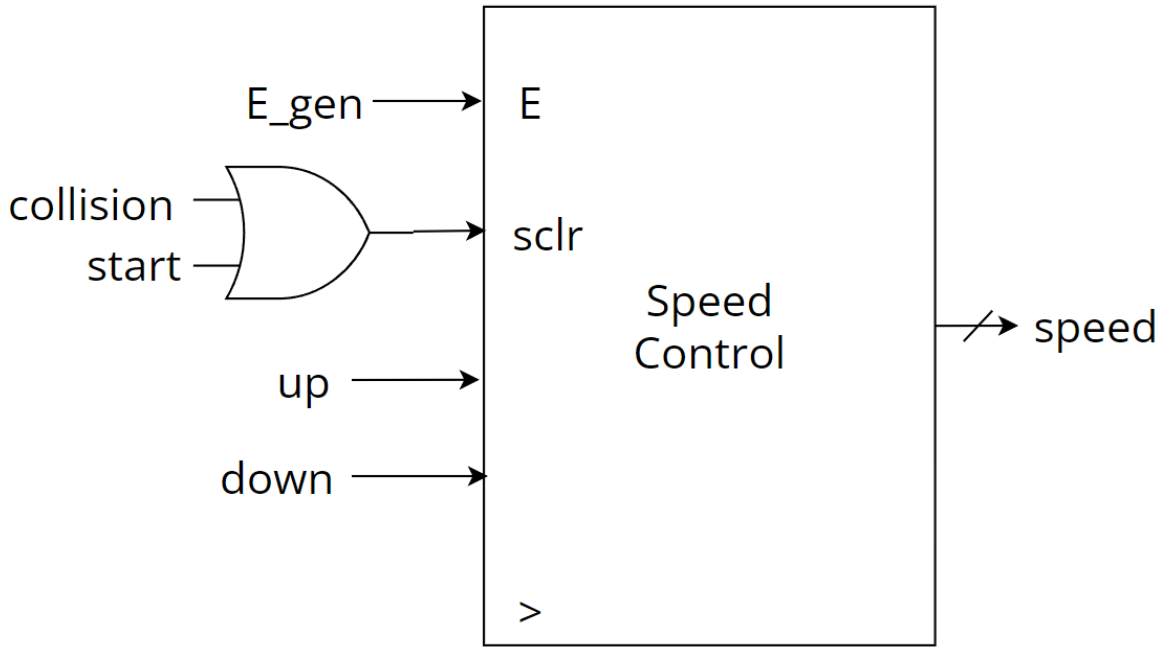
Player Control

- A register that determines the position of the player
- Player uses the left and right buttons to change position
- Takes the game boundaries into account so as not to go too far left/right
- Contains a delay counter so that the player position does not immediately change

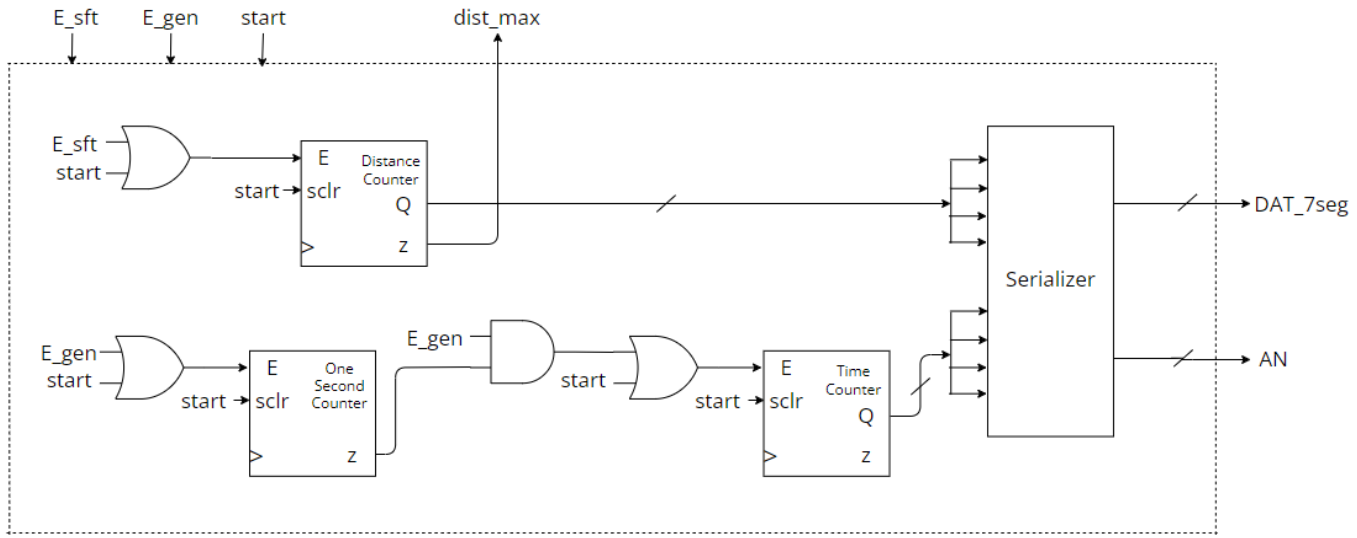


Speed Control

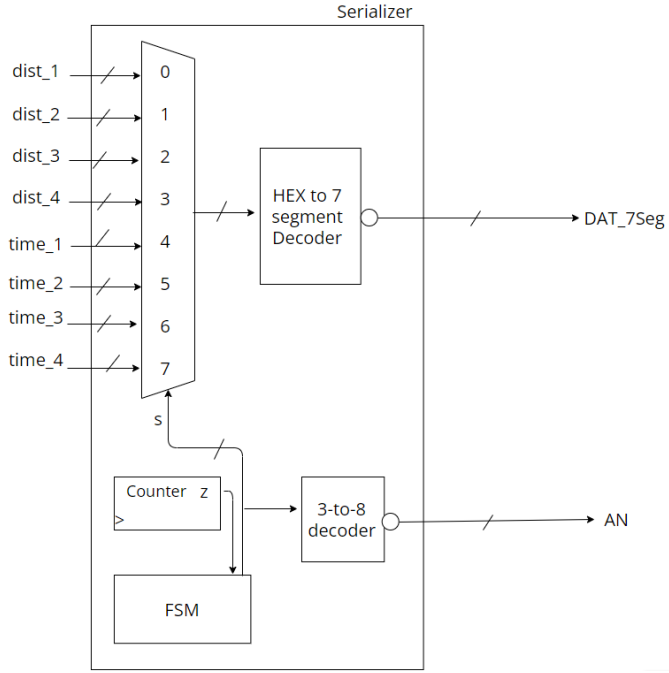
- A register that determines the speed of the player
- Player uses the up and down buttons to increase/decrease speed respectively
- Sets bounds so that the speed does not exceed the value that the maximum number of specified bits would allow or go below zero
- Enable E_gen comes from FSM_Main
- Synchronous clear is determined by an OR operation between collision and start

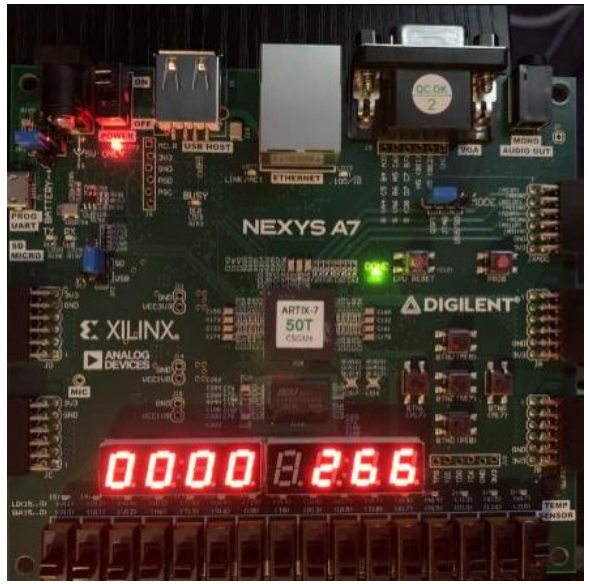


7 Segment Serializer

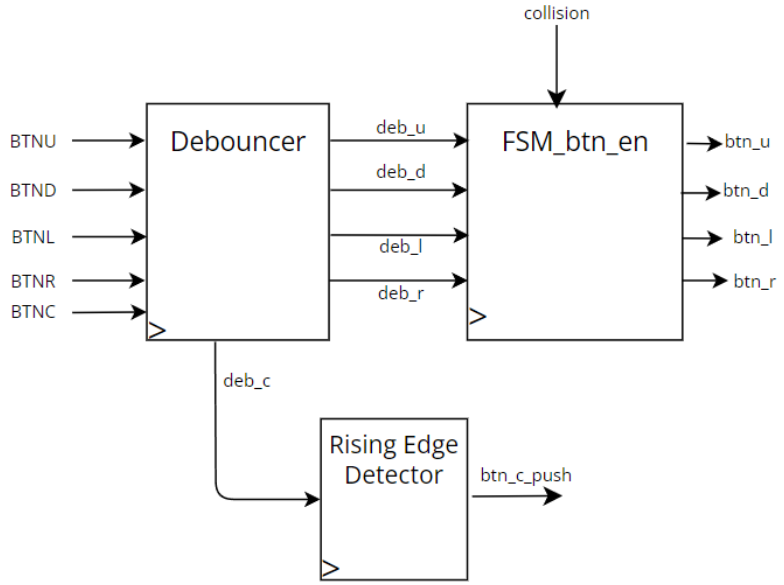


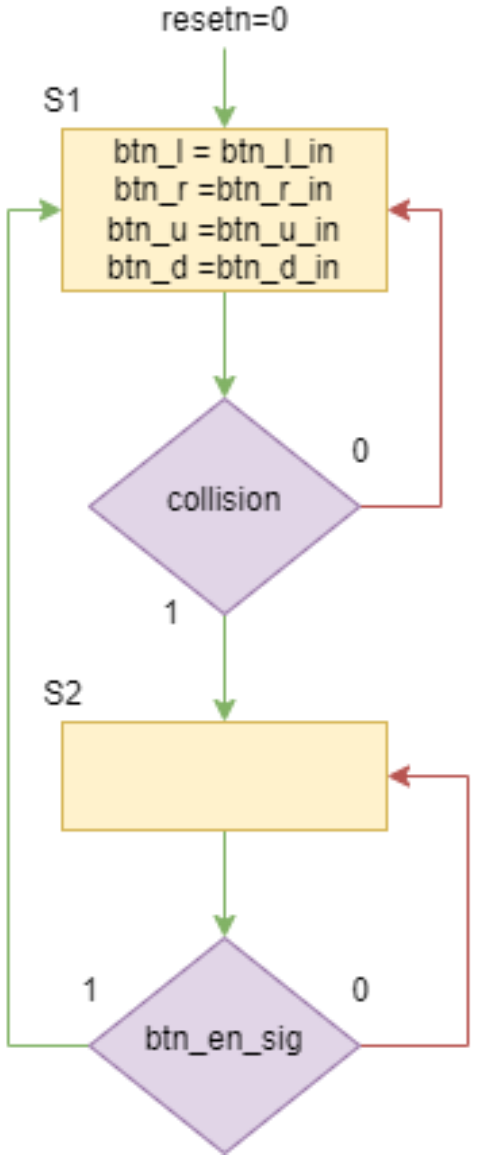
- Embedded with three counters: one second counter, distance counter, time counter
- Serializer receives eight inputs which come from the counters
- Outputs distance on seven segment LED displays
- The 4 left-most LEDs display distance with a max of 9999
- The 3 right-most LEDs display time with a max of 999 seconds





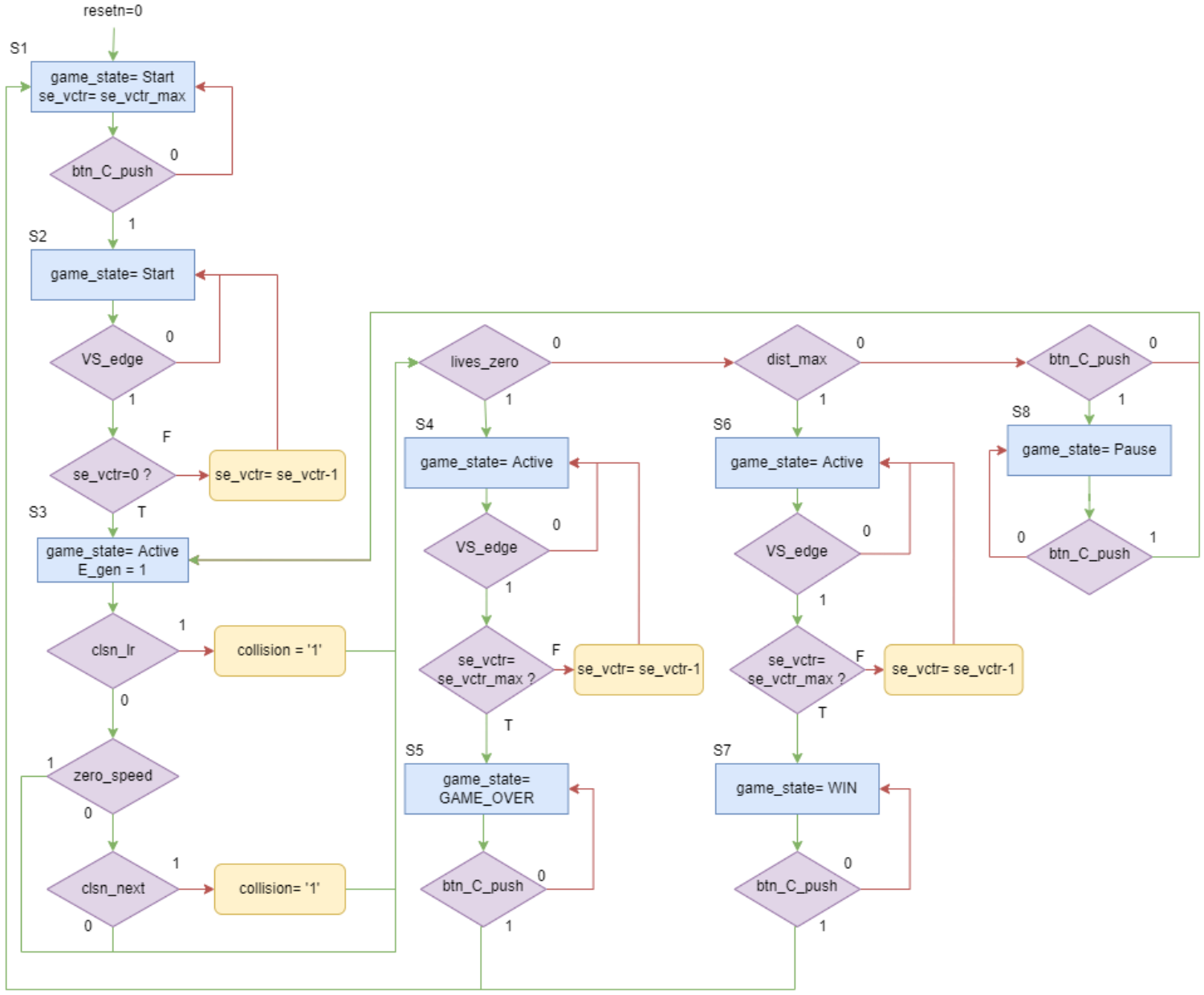
Button Inputs





- The inputs into the rest of the project for the game.
- Each of the 5 buttons was mapped to a debouncer and then the 4 directional buttons are mapped to a small FSM called btn_fsm
- This FSM controls when these button input values would be updated and then passed onto the player and speed control circuits.
- The center button signal gets passed into a rising edge detector for use later in the main FSM

Main FSM


resetn=0
S1
game_state= Start
se_vctr= se_vctr_max
btn_C_push
0
1
S2
game_state= Start
VS_edge
0
1
F
se_vctr=0 ?
se_vctr= se_vctr-1
S3
T
game_state= Active
E_gen = 1
clsn_lr
1
collision = '1'
0
zero_speed
1
0
clsn_next
1
collision= '1'
0
lives_zero
0
1
S4
game_state= Active
VS_edge
0
1
se_vctr=
se_vctr_max ?
F
se_vctr= se_vctr-1
T
S5
game_state=
GAME_OVER
btn_C_push
0
1
dist_max
0
1
S6
game_state= Active
VS_edge
0
1
se_vctr=
se_vctr_max ?
F
se_vctr= se_vctr-1
T
S7
game_state= WIN
btn_C_push
0
1
btn_C_push
0
1
S8
game_state= Pause
0
btn_C_push
1

FSM Main

- The FSM's purpose is to control the game_state variable.
- The game_state is what determines whether the game is paused or playing, and whether you have won or lost.
- It takes in the values zero_lives and dist_max to determine the win conditions, if dist_max=9999 without zero_lives becoming one then you win, you lose if zero_lives becomes one first.
- To continue through any of the menus or pause you click the center d-pad button which controls the btn_c_push signal

Demo