



COL215 – Digital Logic and System Design Department of
Computer Science & Engineering, IIT Delhi, 2025-26

Lab Assignment 8 – P3

By: Preksha Rathal 2024CS50987 & Kesar Shah 2024CS10132

Group3

Contents

1. <u>Introduction</u>	
1.1 Problem Statement	3
2. <u>Design Description</u>	
2.1 Input	3
2.2 Output	3
2.4 Design Decisions	4
3. <u>Verification And Testing</u>	
3.1 Simulation Snapshots	5
3.2 Resource Utilization (Post Synthesis).	5
3.3 Generated Schematics	6
3.4 VGA Monitor Output.	7
4. <u>Constraint Mapping</u>	10
5. Conclusion.....	10

1. Introduction

- Problem Statement

We implement:

1. An **8-bit pseudo-random number generator (LFSR)** to generate a random horizontal spawn position.
2. A **single-port ROM (rival_car_rom)** initialized using *rival_car.coe*.
3. Logic to **randomly initialize rival_x between 44 and 104 pixels** and **rival_y = OFFSET_BG_Y**.
4. A frame-dependent movement system to **move the rival car down slowly**.
5. Collision detection logic to determine if the **rival car overlaps with the main car's bounding box**.
6. Logic to **reset the rival car** after reaching the bottom boundary or after a collision.

2. Design Description

We have added the following inputs and outputs with 8.2

Inputs

Signal	Width	Description
<i>clk</i>	<i>1</i>	<i>100 MHz system clock</i>
<i>reset</i>	<i>1</i>	<i>BTNC – resets the entire game including rival car</i>
<i>hor_pix</i>	<i>10</i>	<i>VGA horizontal pixel counter</i>
<i>ver_pix</i>	<i>10</i>	<i>VGA vertical pixel counter</i>
<i>main_car_x, main_car_y</i>	<i>10 each</i>	<i>Current position of main car</i>
<i>pixel_clock</i>	<i>1</i>	<i>VGA pixel enable</i>

Outputs

Signal	Width	Description
rival_car_rgb	12-bit	Pixel data from rival_car_rom
rival_x, rival_y	Registers	Current top-left position of the rival car
collision_flag	1	1 when rival and main car overlap

100 MHz Clk → VGA_driver → hor_pix, ver_pix, HS, VS, pixel_clock



Display_sprite (Top Module)



bg_rom

main_car_rom



Transparency & Mux Logic



vgaRGB[11:0]

Design Decisions

- **Car Movement:** We used a clock divider to generate a 100 Hz **move_enable** signal, matching the VGA refresh rate. The `car_x` register is updated only when this enable is high and the *FSM* is in a move state. This ensures the car moves smoothly at a constant speed, one update per frame.

- **8-bit Pseudo Random Generator (LFSR)**

We implement an 8-bit LFSR with taps chosen to generate a maximal sequence.

LFSR Characteristics

Width: **8 bits**

Feedback polynomial:

$$\text{new_bit} = q[7] \text{ XOR } q[5] \text{ XOR } q[4] \text{ XOR } q[3]$$

Output: `LFSR_value` (8-bit)

Reset seed: Bitwise XOR of **8 LSBs of both kerberos IDs**

Purpose

The lower bits of the LFSR output are used to compute:

$$\text{random_x} = 44 + (\text{LFSR_value} \% 60)$$

which gives a value between **44 and 104**, matching the allowed road boundaries.

- **Collision Logic:** The boundary checks (`car_x < 244` and `car_x + 14 > 318`) are implemented as combinational logic that determines the `next_state`. This ensures that a collision is detected immediately before the car's position is updated to an invalid value.
- **Button Handling:** The raw `BTNL`, `BTNR`, and `BTNC` inputs are passed through a debouncing module (reused from a previous lab) before being fed into the FSM to prevent erratic state changes from mechanical button bounce.
- **Simulation Parameters:** To make the simulation complete in shorter time, we have decreased the bits for count in the debouncer from 20 bits to 4 bits and the `MOV_DIV` in the `car_fsm` has been changed to 50 for the simulation to work properly.
- **FSM Implementation:** A standard two-block Verilog FSM template was used, as recommended.
 3. A sequential always `@(posedge clk or posedge reset)` block for the state register (`current_state <= next_state`).
 4. A combinational always `@(*)` block to determine `next_state` and update outputs based on `current_state` and inputs.



2. Verification And Testing

- Simulation Snapshots

We have implemented 2 test benches, one to show that a rival car is generated at random points, where we have it start from a lower point on the screen, so that the simulation runs properly, and in which we have changed the x position of the rival car to match the x position of the car so that a collision takes place, and we show that the cars stops moving.

Test Bench 1(tb8.v): Collision with main car

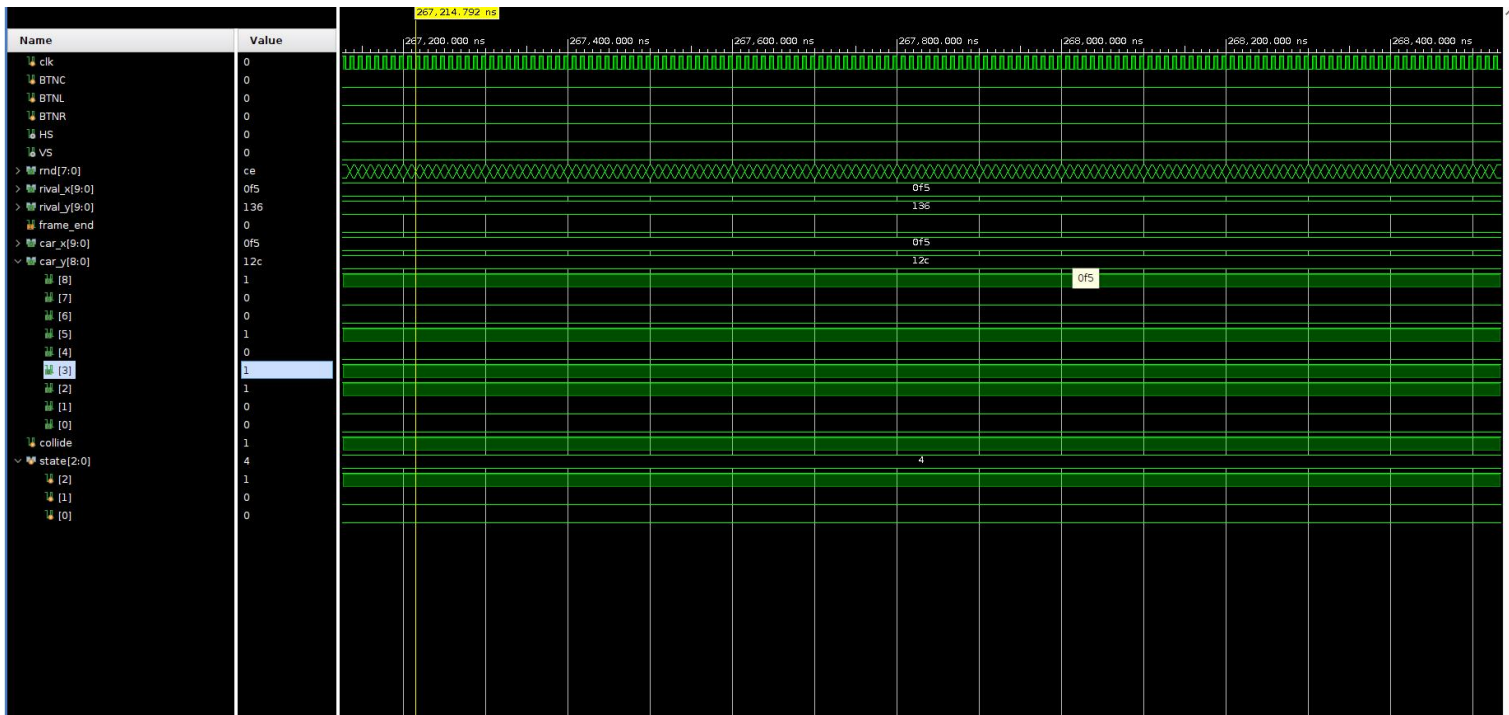


Fig. collision with rival car, state changes to state 4 and both cars stop moving even when time is progressing

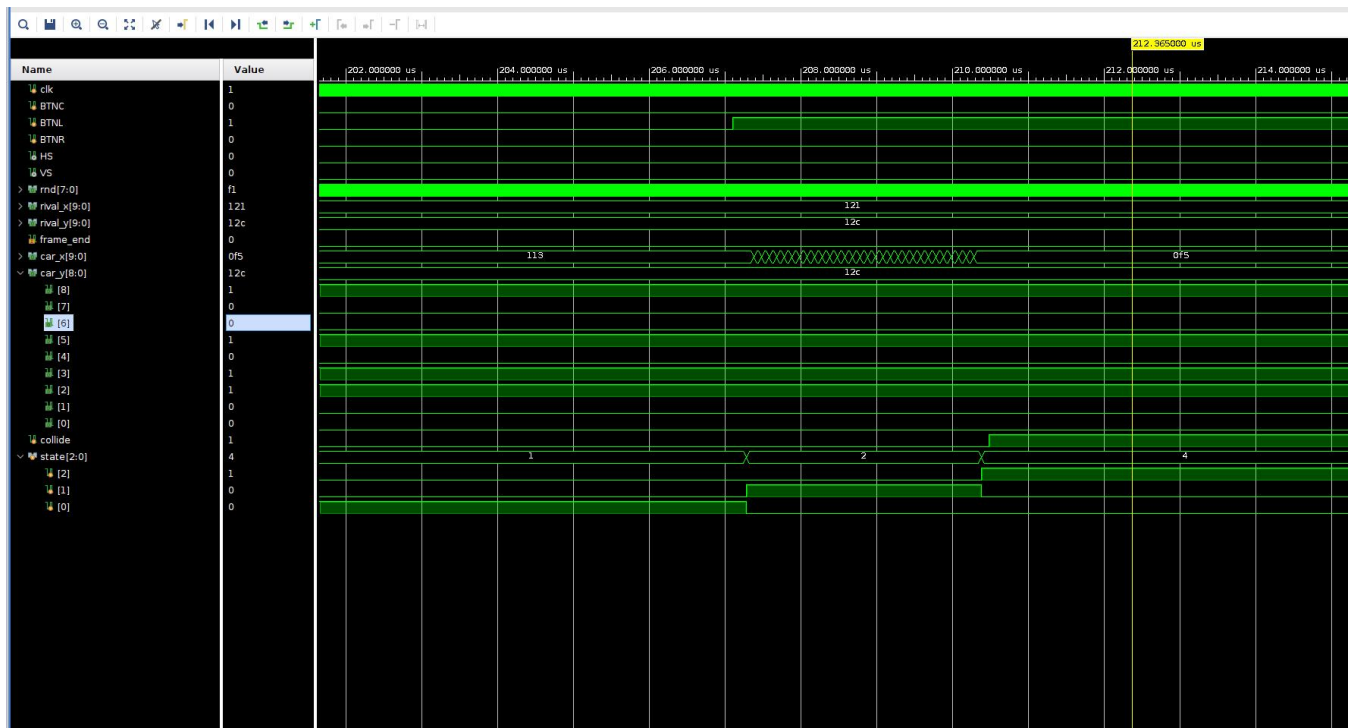


Fig. Both cars have the same y position(12c) and x coordinates are close enough ($x_{car} = 121$, $x_{rival_Car} = 113$) ie less than the width of the car and hence a collision takes place

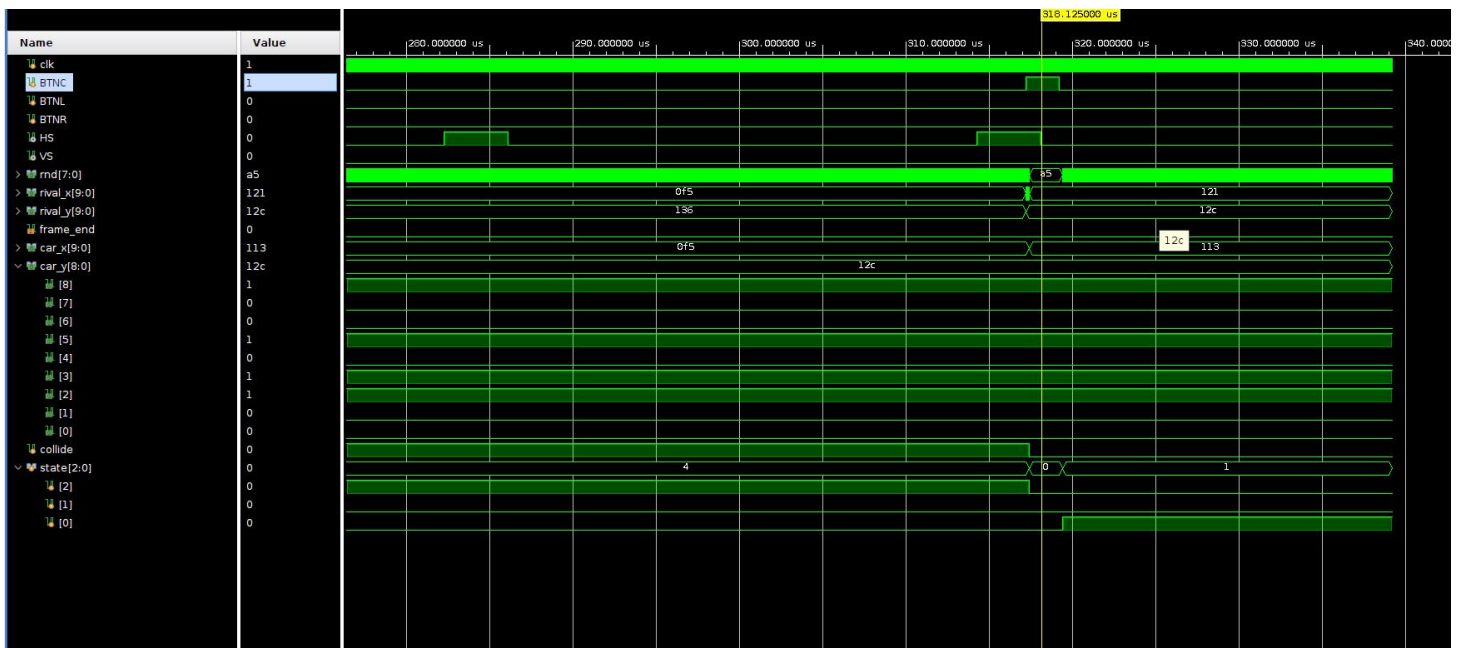


Fig. Pressing BTNC after collision causes respawning of rival car (with new coordinates) and state becomes START

Testbench 2: Showing random generation of rival car coordinates

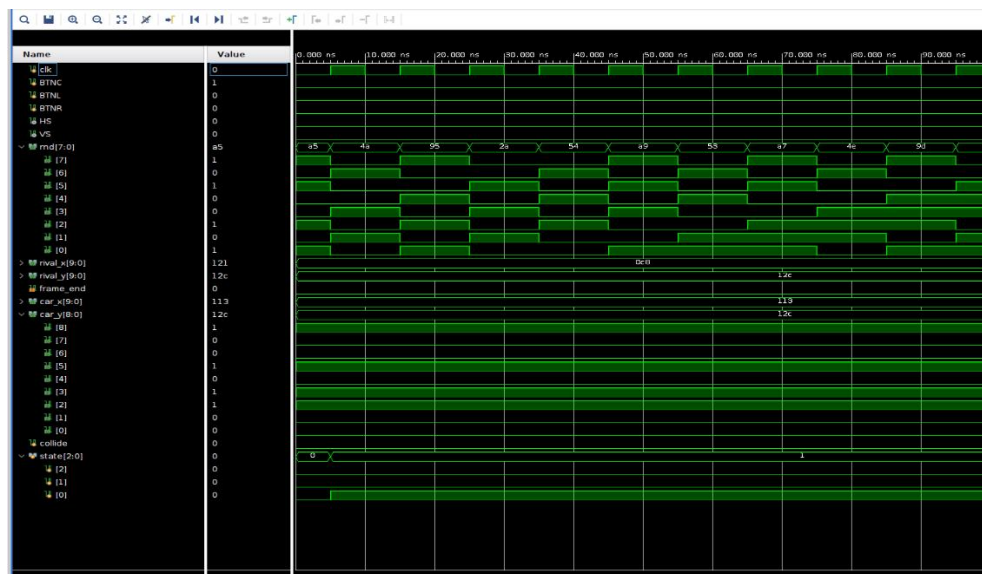


Fig. Random values generated using Linear Feedback Shift Register

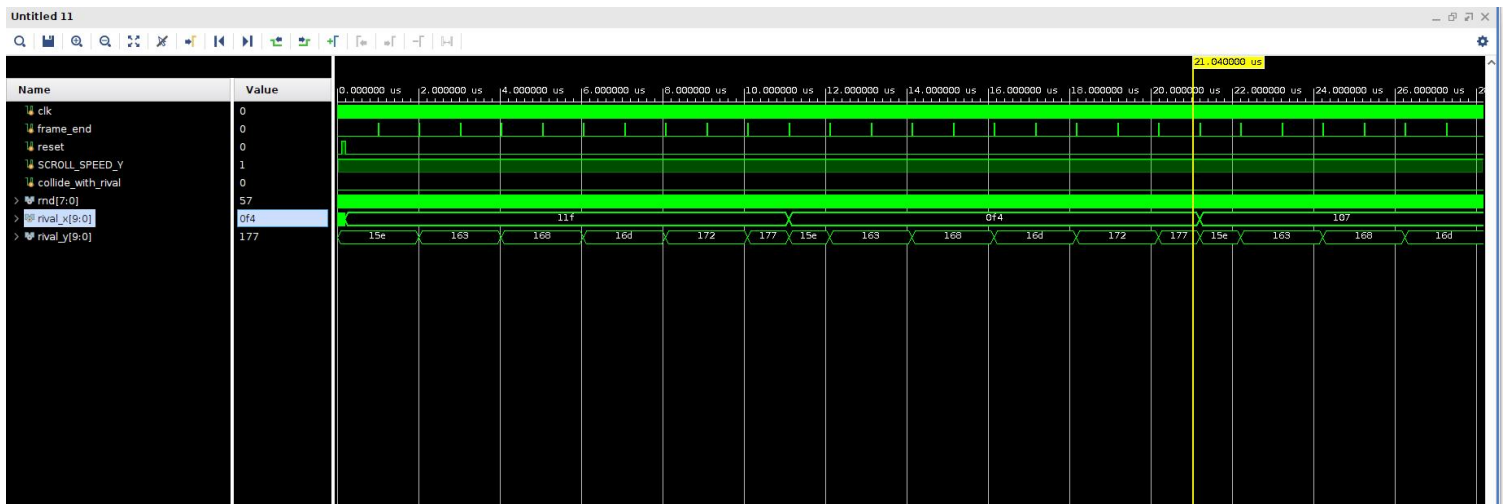


Fig. y coordinate for rival car starts at 15c, x coordinate is randomly generated

Y coordinate continuously increases till 177 then the car respawns at 15c again with different x coordinate ie randomly generated using Linear Feedback Shift Register

X coordinate (initial) =11f

X coordinate (2nd gen) = 0f4

- Resource Utilization (Post Synthesis)

- DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	90	0.00

DSPs: 90 (col length:60)

BRAMs: 100 (col length: RAMB18 60 RAMB36 30)

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	654	0	0	20800	3.14
LUT as Logic	654	0	0	20800	3.14
LUT as Memory	0	0	0	9600	0.00
Slice Registers	311	0	0	41600	0.75
Register as Flip Flop	303	0	0	41600	0.73

Register as Latch	8	0	0	41600	0.02	
F7 Muxes	0	0	0	16300	0.00	
F8 Muxes	0	0	0	8150	0.00	
+-----+-----+-----+-----+-----+-----+						

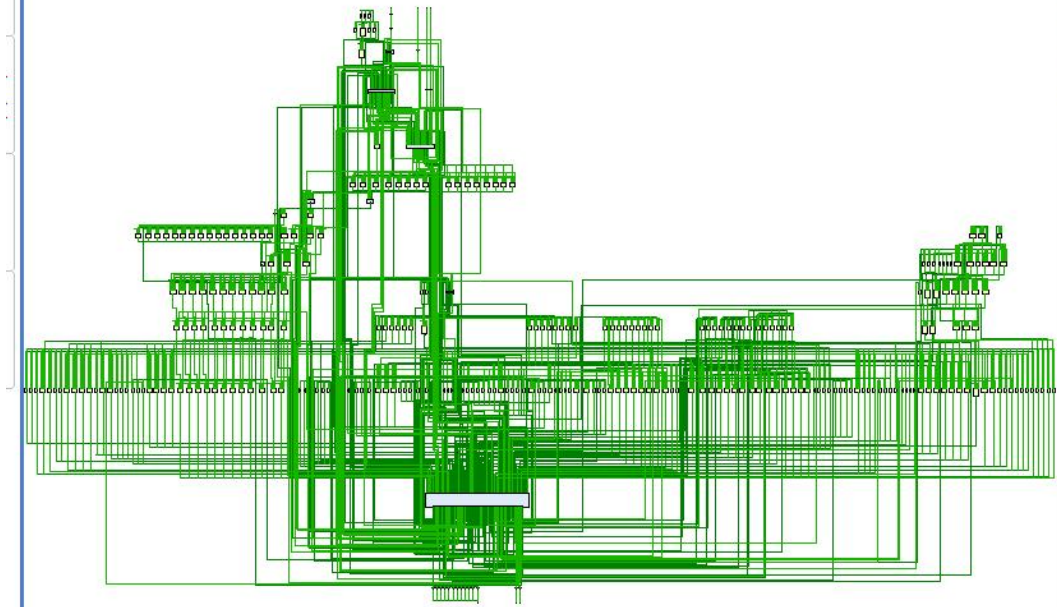
2. Flip Flops

Register as Flip Flop	303	0	0	41600	0.73	
-----------------------	-----	---	---	-------	------	--

Report Cell Usage:

+-----+-----+-----+		
	Cell	Count
+-----+-----+-----+		
1	bg_rom	1
2	main_car_rom	1
3	rival_car_rom	1
4	BUFG	2
5	CARRY4	150
6	LUT1	53
7	LUT2	296
8	LUT3	120
9	LUT4	191
10	LUT5	89
11	LUT6	132
12	FDCE	18
13	FDPE	12
14	FDRE	265
15	FDSE	8
16	LDC	8
17	IBUF	4
18	OBUF	14
+-----+-----+-----+		

• Generated Schematics



- VGA Monitor Output

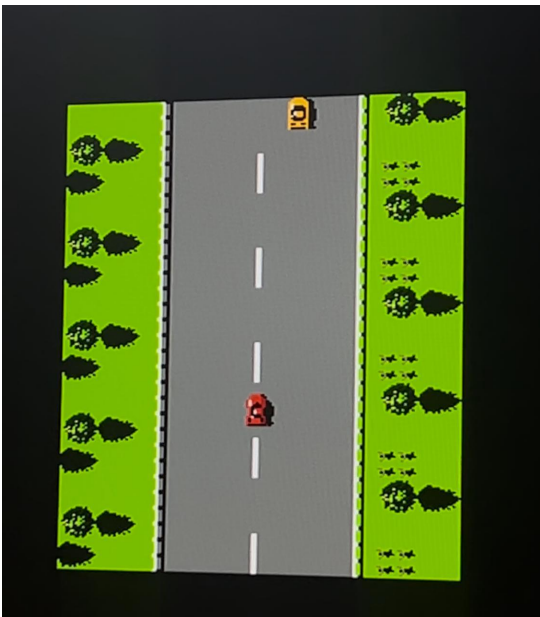
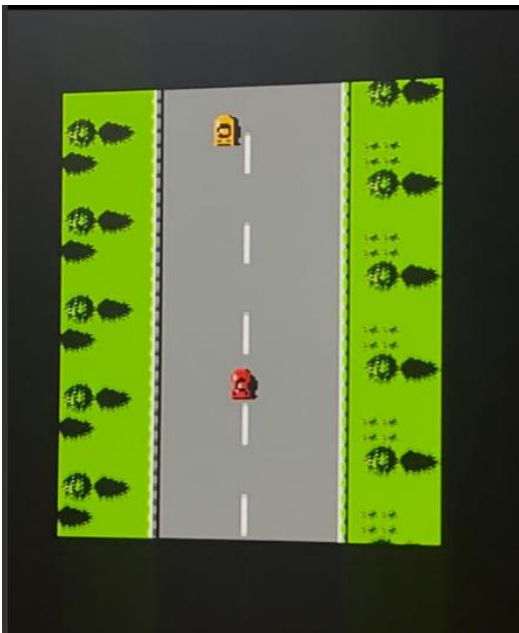


Fig. Random generation of car at right side
 Random generation of car at left
 of road
 side of the road

Fig.

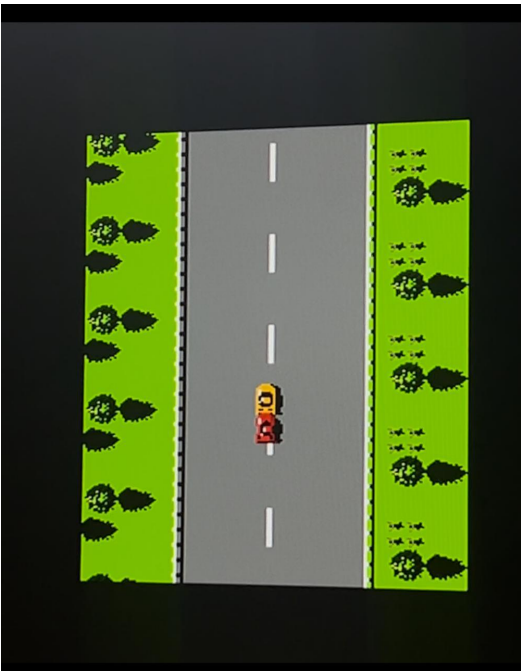


Fig. Collision of car with rival car causing
Collision of car with boundary also
Stopping of game
causes stopping of game

Fig.

5. Constraint Mapping

Signal	Pin	Description
clk	W5	100 MHz System Clock
vgaRGB[11]	G19	Red Channel MSB (R3)
vgaRGB[10]	H19	Red Channel Bit 2 (R2)
vgaRGB[9]	J19	Red Channel Bit 1 (R1)
vgaRGB[8]	N19	Red Channel LSB (R0)
vgaRGB[7]	J17	Green Channel MSB (G3)
vgaRGB[6]	H17	Green Channel Bit 2 (G2)
vgaRGB[5]	G17	Green Channel Bit 1 (G1)
vgaRGB[4]	D17	Green Channel LSB (G0)
vgaRGB[3]	N18	Blue Channel MSB (B3)
vgaRGB[2]	L18	Blue Channel Bit 2 (B2)
vgaRGB[1]	K18	Blue Channel Bit 1 (B1)
vgaRGB[0]	J18	Blue Channel LSB (B0)
HS	P19	Horizontal Sync Signal
VS	R19	Vertical Sync Signal

6. Conclusion

The rival car module was successfully integrated with the existing VGA-driven game. All required features were implemented:

- A correct 8-bit LFSR to randomize spawn locations
- A BRAM-based single-port ROM for sprite storage
- Smooth vertical movement using frame-based timing
- Correct hitbox-based collision detection
- Automatic respawn logic for normal gameplay

The design works reliably on both simulation and Basys3 hardware, completing all tasks outlined in Part III of the assignment.