COL215- ASSIGNMENT 8 Part 3
Road Fighter Game

COL- 215

HARSH VERMA	PARTH RATHI ARVIND
2024CS10499	2024CS50875

Table of Contents

1. Aim / Introduction

The aim of Part 3 is to extend the Road Fighter game by introducing a moving rival car and implementing collision detection between the main car and this rival. A pseudo-random number generator (LFSR) is used to select different horizontal spawn positions for the rival car, making gameplay less predictable. The rival car moves downward continuously using frame-based timing derived from the VGA vertical sync, and the system checks for overlap between the two cars to detect collisions. This part builds on the previous modules and adds dynamic game behaviour entirely through hardware logic.

2. Design and Implementation

Top Module: Display_Sprite

- The Display_sprite module integrates the VGA controller, sprite ROMs, FSM logic, rival car logic and background scroll mechanism.
- The main car sprite (14×16 pixels) is drawn at a fixed vertical position car_y = 300 and a variable horizontal position car_x, which is controlled by the FSM.
- The rival car starts from the top(bg y offset) with the x coordinate generated pseudo randomly using linear feedback shift register with the seed taken from our entry number.
- A background scroll effect is achieved by incrementing a vertical offset (bg_y_offset) periodically. This is implemented using a counter scroll_counter and a scroll period parameter.

FSM Module: fsm_car_state

The FSM is responsible for controlling the car's horizontal movement, collision detection, and reset behavior. It uses five states, each encoded as a 3-bit signal:

- START (0) : Initializes the car position upon the beginning, that is, after reset. The car is initialized to 270.
- IDLE (1) : No operation is carried out. The car is stationary.
- COLLIDE (2) : This state is achieved when collision of car occurs with the road boundaries. The background stops scrolling. Reset has to be used to restart the game. The car also detects collision with the rival car.
- LEFT_CAR (3) : When BTNL is pressed, the car moves horizontally left. It uses a counter to make sure that multiple movements are not made. Move step is 1 pixel.
- RIGHT_CAR (4) : When BTNR is pressed, the car moves horizontally left. It uses a counter to make sure that multiple movements are not made. Move step is 1 pixel.

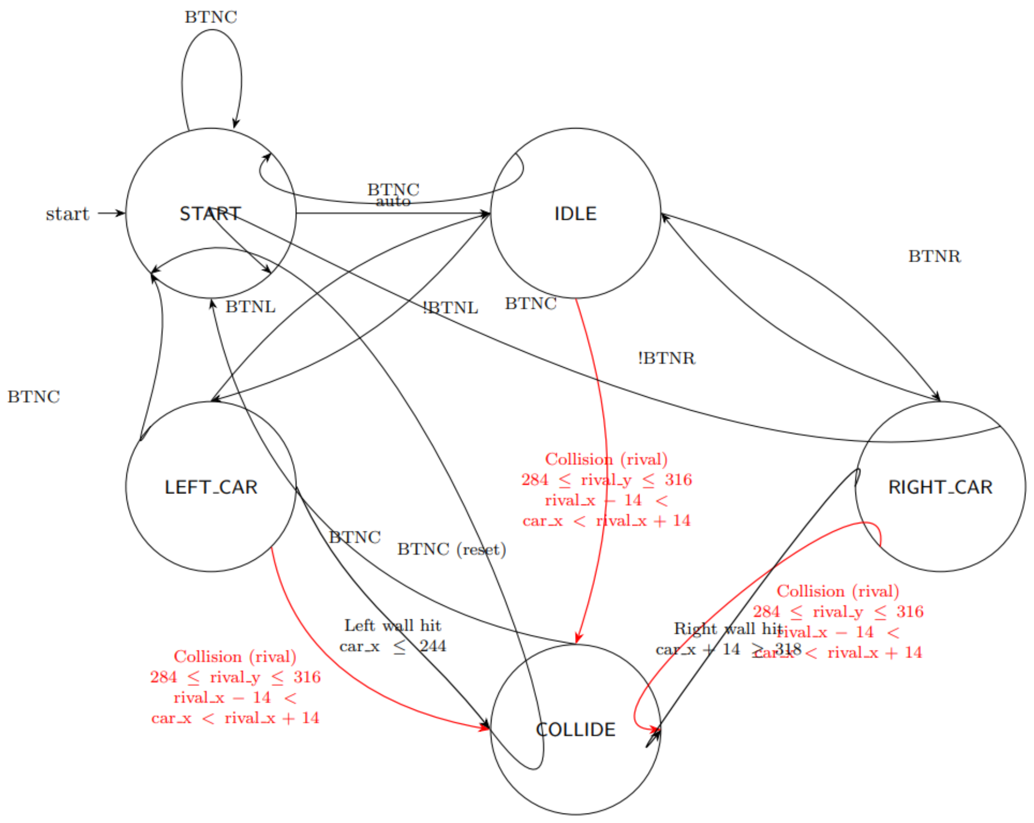


Figure 1 FSM STATE DIAGRAM

Note that collision state here includes collision with the rival car as well.

Sub Module: Debouncer

The reset push button is debounced with a counter-based filter to avoid spurious triggers. Signal changes in very short interval will be discarded.

Sub Module: lfsr

The LFSR generates an 8-bit pseudo-random value used to choose the rival car's horizontal spawn position. It works by shifting its register each clock cycle and inserting a feedback bit formed by XORing taps (d7, d5, d4, d3). The register resets to a fixed SEED whenever the game restarts. Its compact design produces fast, hardware-friendly randomness with minimal logic. The seed is based on the xor of the 8 lsbs of our entry number.

Some decisions with respect to the design

- Movement, debounce, and scroll speeds are parameterized for easy tuning.
- Move step in the code is 1 pixel. In the simulation it is 5 pixel.
- For simulation several counters were changed from the hardware code. Scroll_period to 30 from 500000.

Move_Period to 2 from 2000000
Debounced_limit to 10 from 1000000
Lfsr seed from 8’h98 to 8’h48 (original seed gave collision first, so avoiding)

- Rival car moves 1 pixel down after every frame in the hardware. But in the simulation we increased it to 20; else it would have to run for a long time.
- Rival car’s x coordinate is generated pseudo randomly using lfsr
- Each button has a separate debouncer to ensure reliable input detection.
- Background scrolling pauses during collision and resets at game start.
- Continuous press of BTNR or BTNL allows smooth movement — the car keeps moving without repeated button presses. We have used a counter to control the speed (2000000 cycles)
- BTNC acts as a global reset to restart the game anytime. After collision the game can only restart after pressing BTNC.
- The car’s vertical position is fixed to simplify movement and collision logic. But background is scrolled for the movement effect. Scroll period is 500000.

3. Testbench and Simulation

The testbench verifies the following scenarios:

- Shows right and left movement.
- Shows collision with the rival car
- Shows randomly genrated postion(x cordinate) of the rival car
- Shows fsm state of the car
- Shows main car position and rival car position.

.

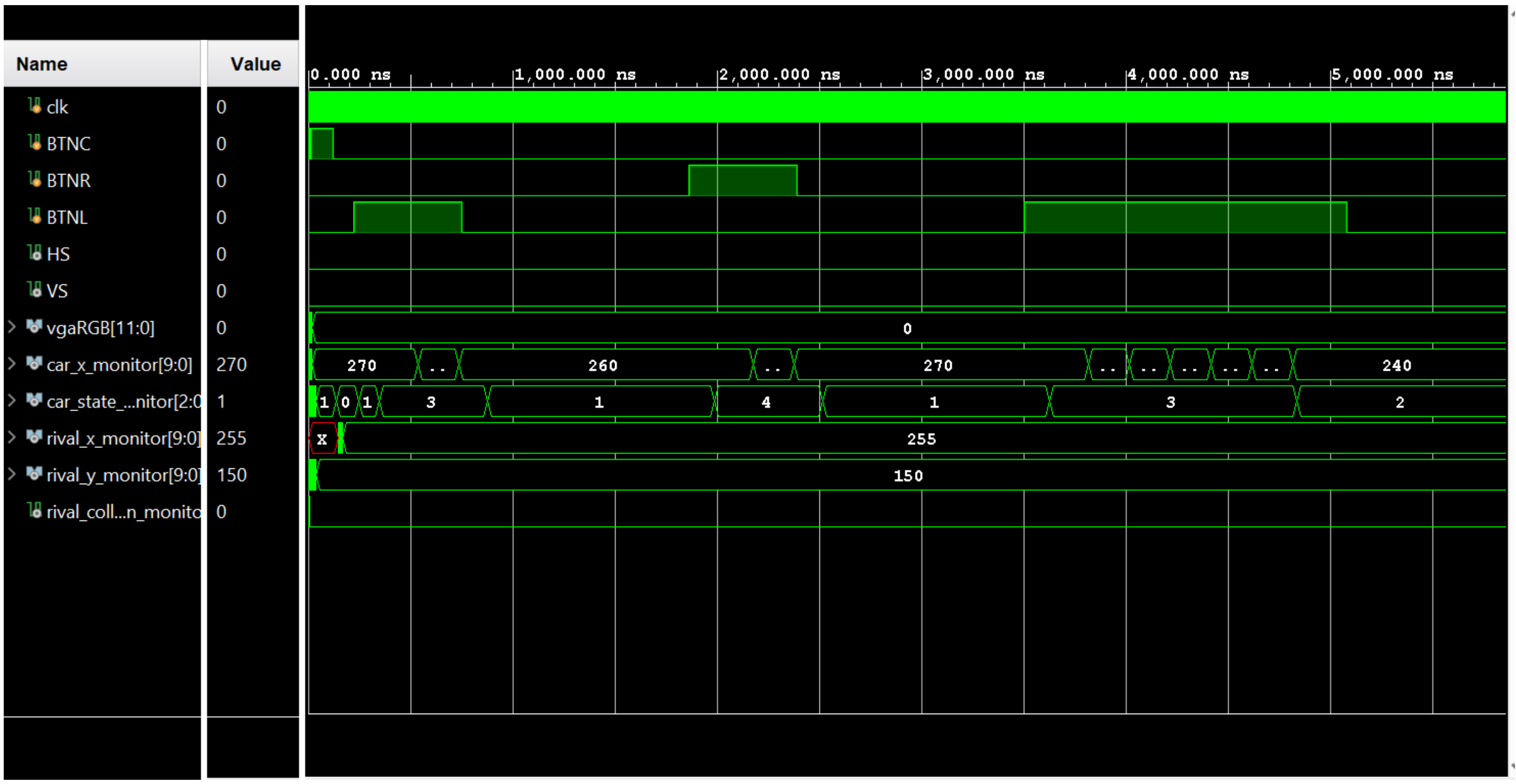


Figure 2 Shows initial left and right movement of the main car. Also shows collision with walls. Shows position of the rival car.

Note: fsm states Start=0 Idle=1 Collide=2 Left_car=3 Right_car=4

Left Collision boundary condition – car_x < 244

Right Collision Boundary Condition – car_x > 304

Move Step for simulation is 5 pixels.

Move Step for hardware is 1 pixel.

Rival Car vertical step is 1 pixel per frame in hardware.

Rival Car vertical step is 20 pixel per frame in simulation.

Horizontal position of the main car is fixed and it is 300.

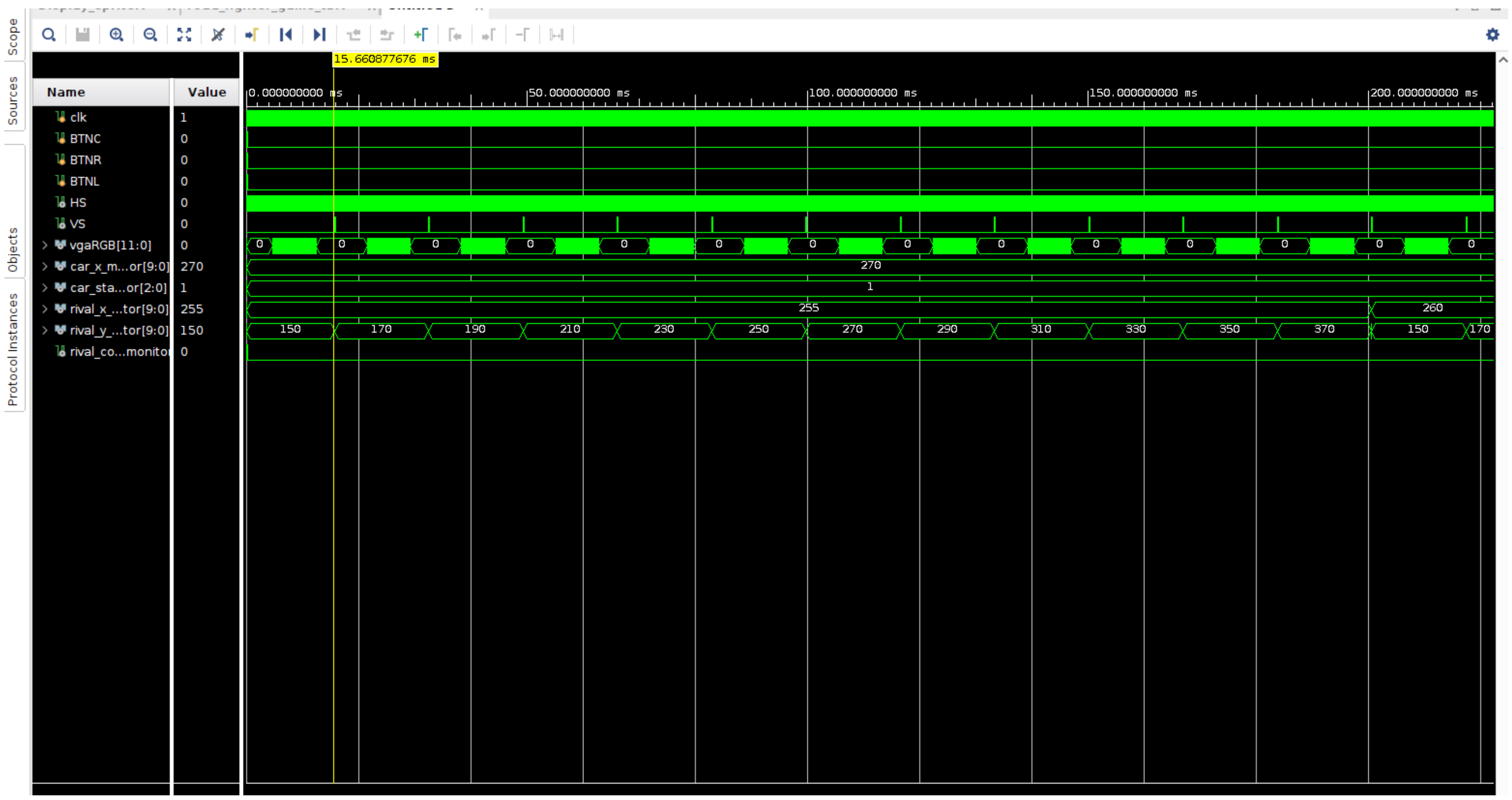
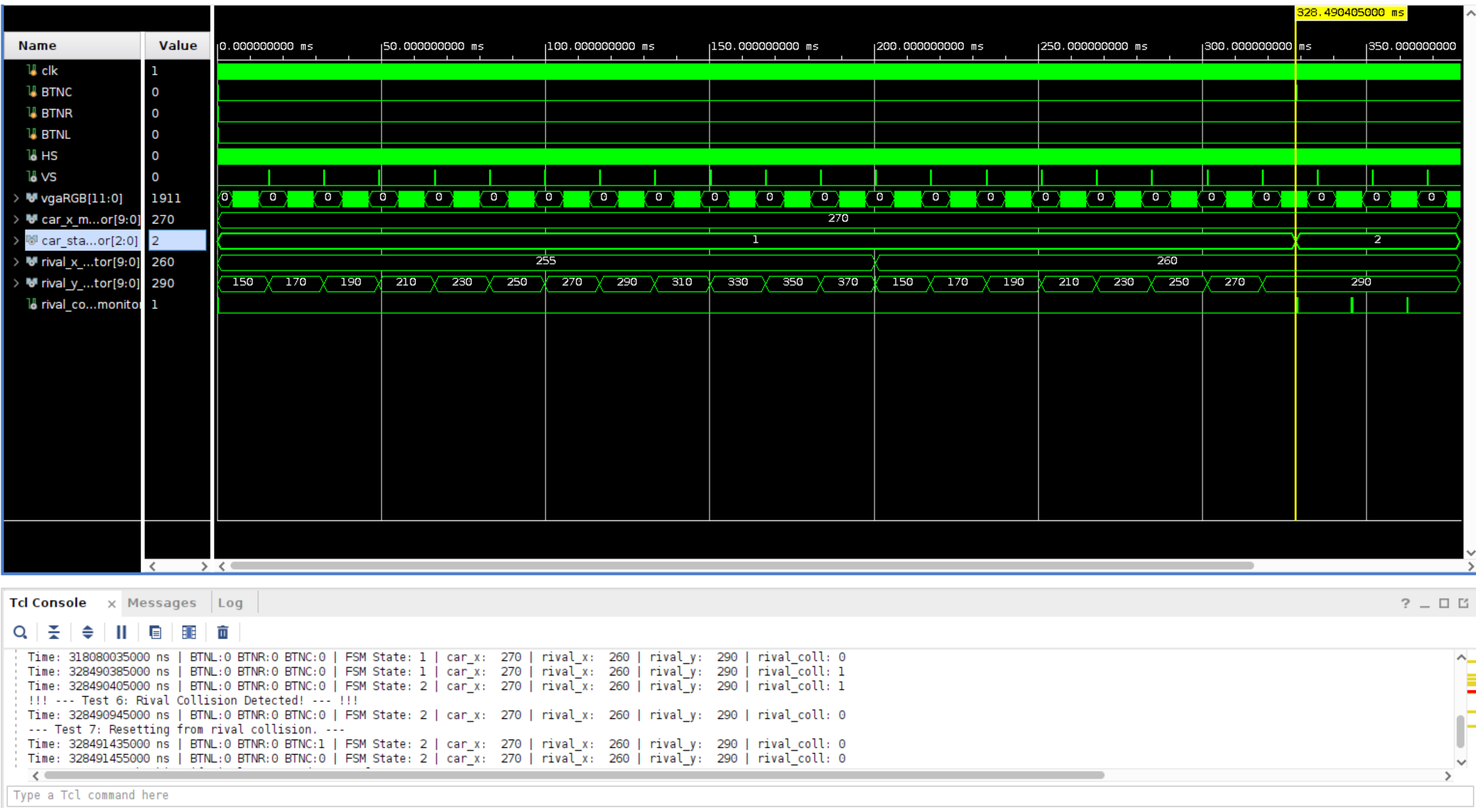


Figure 3 Shows position of the rival car. Spawns first at x=255. After reaching bottom without collision(y cord=390) ; spawns at new position (random) x=260



```
Time: 318080035000 ns | BTNL:0 BTNR:0 BTNC:0 | FSM State: 1 | car_x:  270 | rival_x:  260 | rival_y:  290 | rival_coll: 0
Time: 328490385000 ns | BTNL:0 BTNR:0 BTNC:0 | FSM State: 1 | car_x:  270 | rival_x:  260 | rival_y:  290 | rival_coll: 1
Time: 328490405000 ns | BTNL:0 BTNR:0 BTNC:0 | FSM State: 2 | car_x:  270 | rival_x:  260 | rival_y:  290 | rival_coll: 1
!!! --- Test 6: Rival Collision Detected! --- !!!
Time: 328490945000 ns | BTNL:0 BTNR:0 BTNC:0 | FSM State: 2 | car_x:  270 | rival_x:  260 | rival_y:  290 | rival_coll: 0
--- Test 7: Resetting from rival collision. ---
Time: 328491435000 ns | BTNL:0 BTNR:0 BTNC:1 | FSM State: 2 | car_x:  270 | rival_x:  260 | rival_y:  290 | rival_coll: 0
Time: 328491455000 ns | BTNL:0 BTNR:0 BTNC:0 | FSM State: 2 | car_x:  270 | rival_x:  260 | rival_y:  290 | rival_coll: 0
```

Figure 4 Shows collision of main car and rival car(yellow line)

Note that to run the simulation fast we increased vertical downward movement of rival car to 20 pixels per frame. That's why y coordinate of rival car is not exactly 300 (which is same as of the main car) but stuck at 290, because 300 will be covered in the next step. The same is reflcted in the log.

Fsm state of the main car changes to 2 post collision and no change in the position both the cars after that(as required)

4. Constraints and Hardware Testing

The design was synthesized on the **Basys3 FPGA** board. VGA Driver was used to display the game on the lab desktop.

During testing, the following were verified:

- Background was scrolling as long as there was no collisions.
- Using BTNR for right movement.
- Using BTNL for left movement
- Using BTNC for reset.
- Collision detection with the road boundaries and the rival car.
- Rival car's x coordinate was generated randomly.

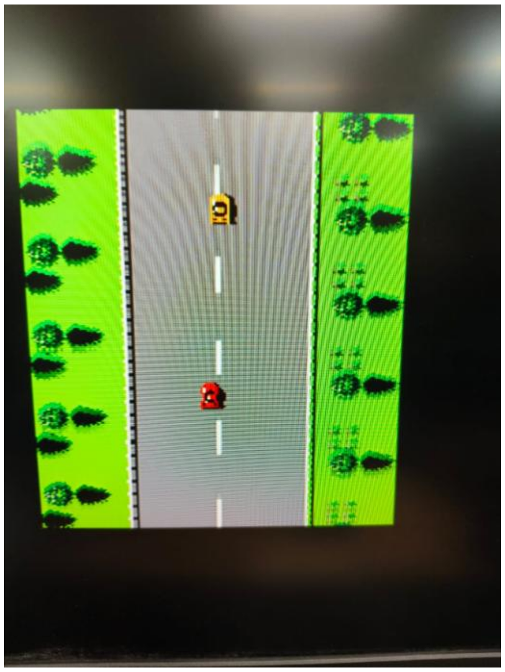


Figure 5 Start

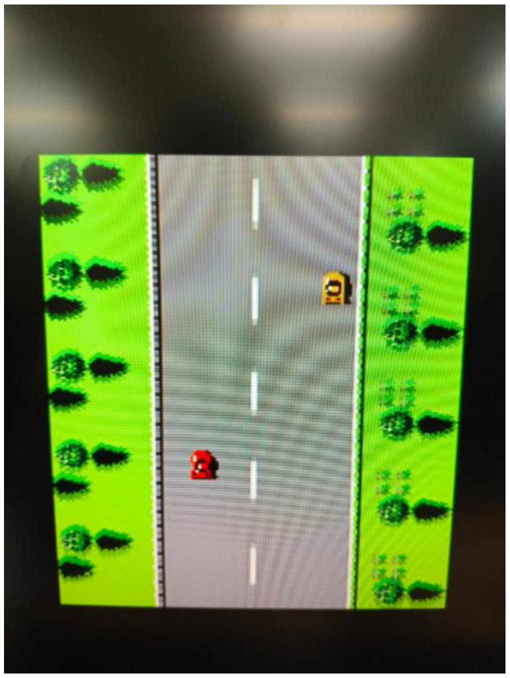


Figure 6 Another position of rival car

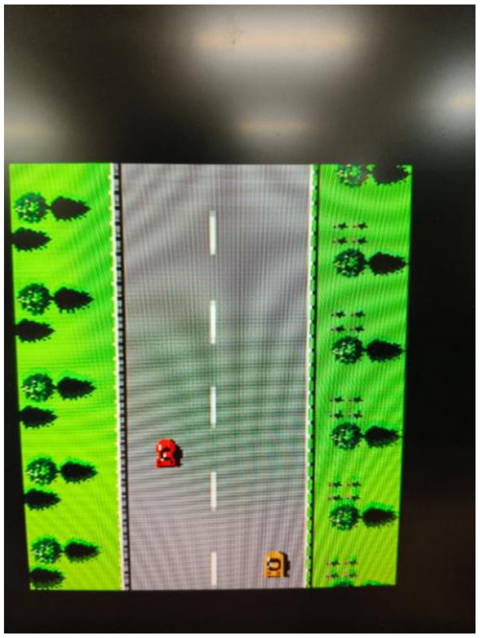


Figure 7 Passed without collision

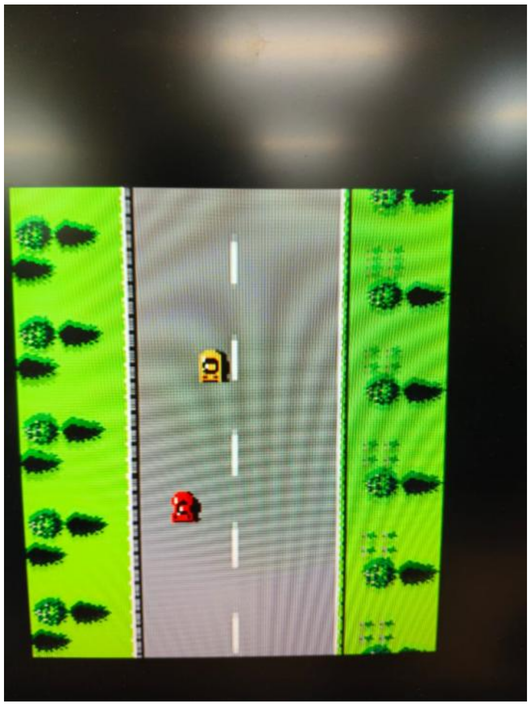
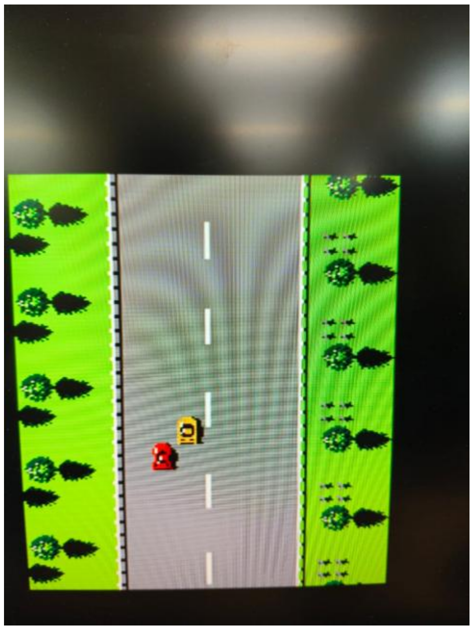


Figure 8 New position of rival car



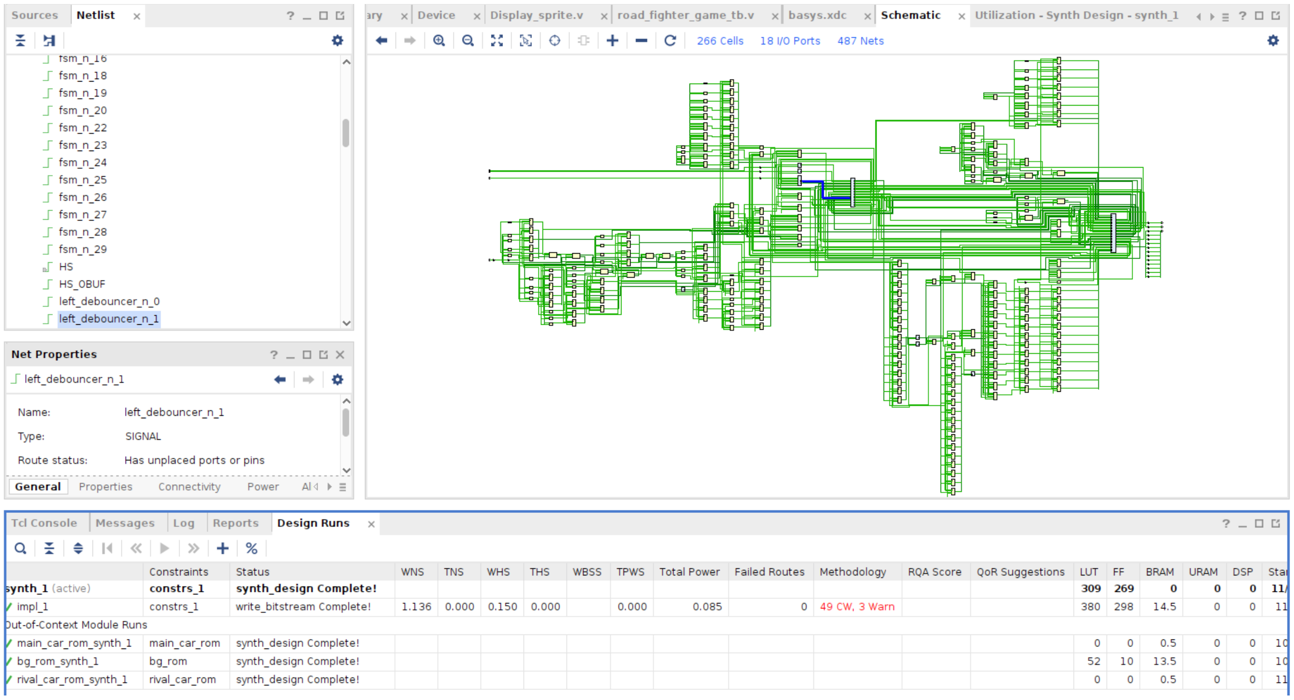
Shows collision of the main car and rival car.

5. Resource Utilization

Resource utilization after synthesis and implementation.

- No Block RAMs or DSP units are used, showing that the design relies purely on lightweight logic and distributed memory.
- Flip-flop usage increases only slightly, reflecting the extra registers for LFSR, rival car position, and frame counters.
- The total utilization stays far below FPGA limits, meaning the design is efficient, scalable, and leaves ample space for further game features.

Resource Type	Used	Available	Utilization (%)
LUTs	309	20800	1.49
Block RAMs	0	50	0.00
DSPs	0	90	0.00
Flip Flops	269	41600	0.65



Name: left_debouncer_n_1

Type: SIGNAL

Route status: Has unplaced ports or pins

	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Sta
ynth_1 (active)	**constrs_1**	**synth_design Complete!**												**309**	**269**	**0**	**0**	**0**	**11/**
✓ impl_1	constrs_1	write_bitstream Complete!	1.136	0.000	0.150	0.000		0.000	0.085	0	49 CW, 3 Warn			380	298	14.5	0	0	11
ut-of-Context Module Runs																			
✓ main_car_rom_synth_1	main_car_rom	synth_design Complete!												0	0	0.5	0	0	10
✓ bg_rom_synth_1	bg_rom	synth_design Complete!												52	10	13.5	0	0	10
✓ rival_car_rom_synth_1	rival_car_rom	synth_design Complete!												0	0	0.5	0	0	11

Figure 9 Luts and flip flops

6. Schematics

The schematic accurately represents the interconnection between the FSM, debouncers, VGA driver, lsfr and ROM modules. It clearly illustrates signal flow from user inputs to visual output on the VGA display.

Note: Schematics is shown on next page(see below) for better clarity.

7. Conclusion

Part 3 successfully extends the Road Fighter game by adding a moving rival car with random spawning and collision detection. The integration of an LFSR-based random generator, a frame-controlled movement system, and FSM collision handling works reliably in hardware. The rival car behaves unpredictably, moves smoothly down the track, and correctly triggers the COLLIDE state on impact. Overall, the design remains modular, efficient, and fully compatible with the earlier parts of the game. This design can be extended to add new features and converted into an actual game.