COL215- ASSIGNMENT 8 Part 2

Road Fighter Game

COL- 215

HARSH VERMA	PARTH RATHI ARVIND
2024CS10499	2024CS50875

Table of Contents

1. Aim / Introduction

The objective of this part is to design and implement a Finite State Machine (FSM) that controls the horizontal movement of the main car in the Road Fighter game and detects collisions with the road boundaries. This module builds upon the Road Fighter game developed in Part I, extending it with interactive control logic. The car's left and right movements are driven by the BTNL and BTNR push buttons on the Basys 3 board, while the BTNC button serves as a reset control to restart the game after a collision.

2. Design and Implementation

Top Module: Display_Sprite

- The Display_sprite module integrates the VGA controller, sprite ROMs, FSM 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.
- 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.
- 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.

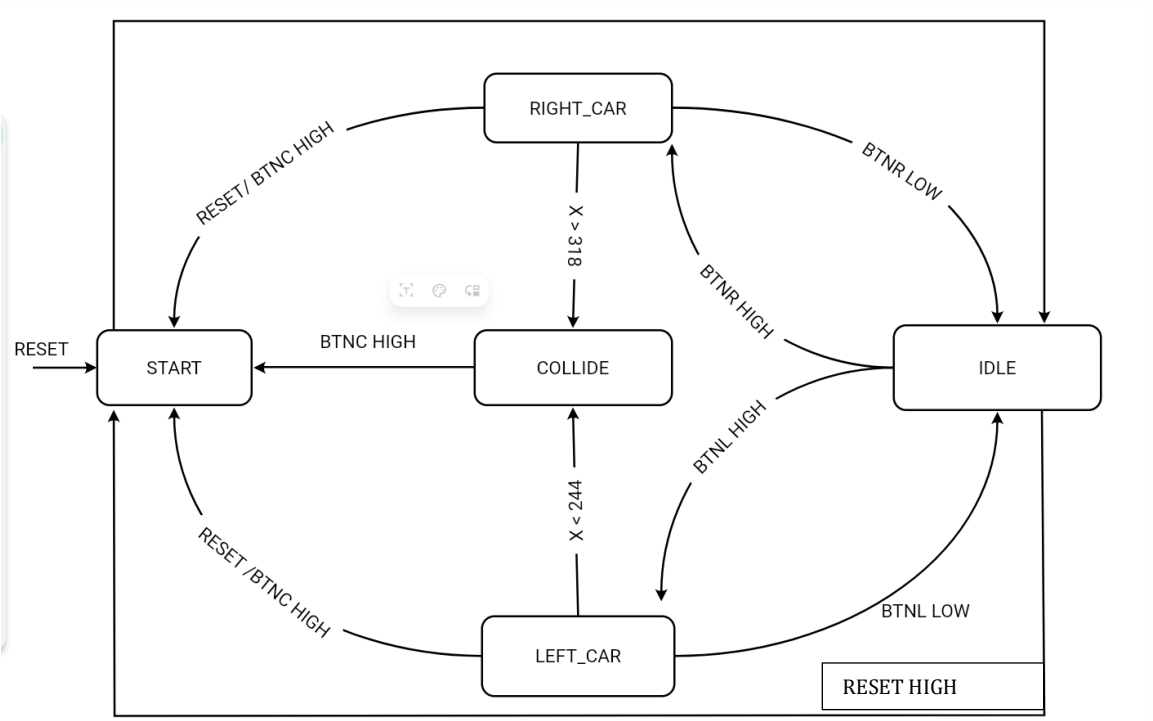

Figure 1 FSM STATE DIAGRAM

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.

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

- 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 right and left movement
- Shows fsm state of the car
- Shows car position

.

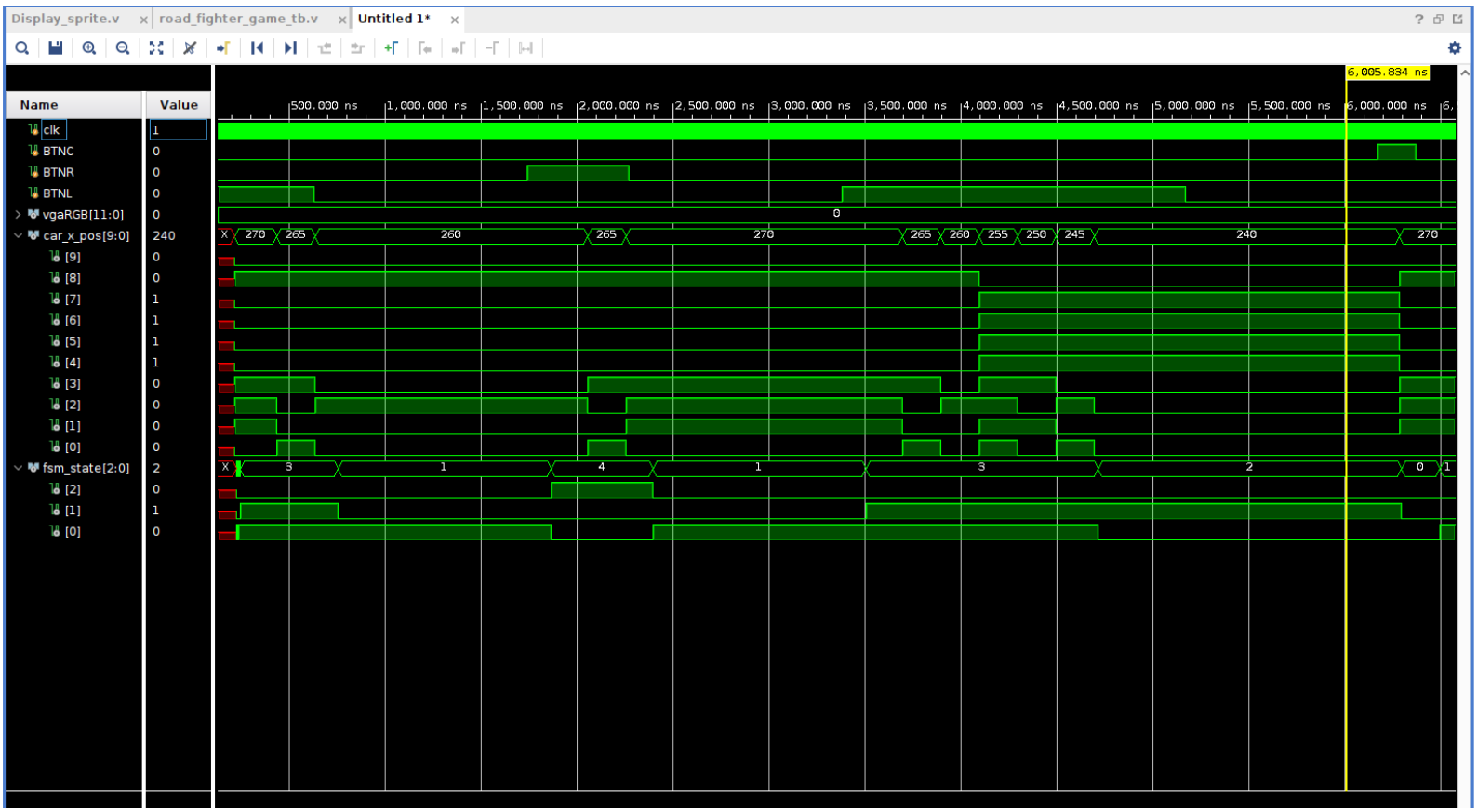

Figure 2 Shows left and right movement of the car. Also shows collision with left wall(yellow line)

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.

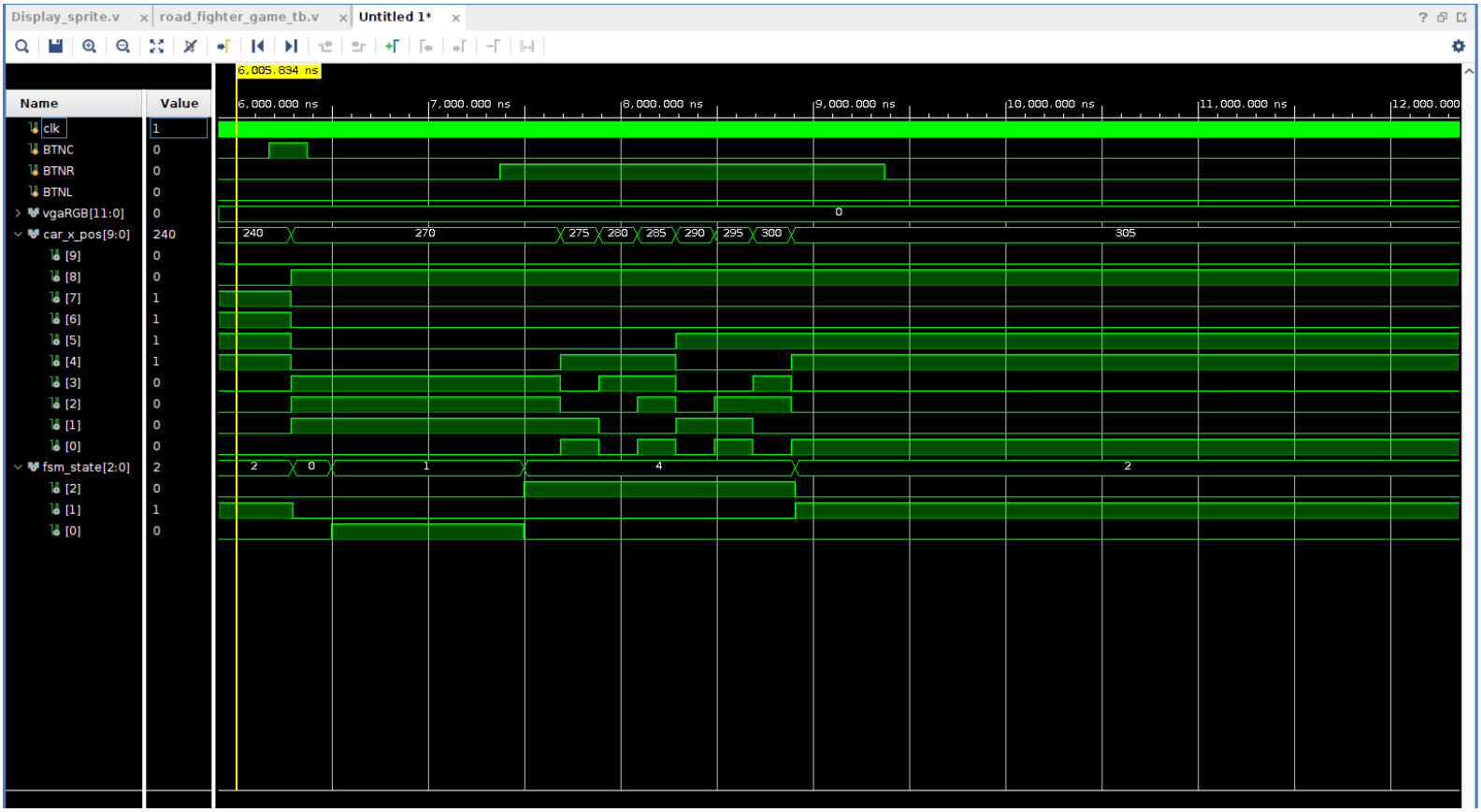

Figure 4 Shows right collision at 305

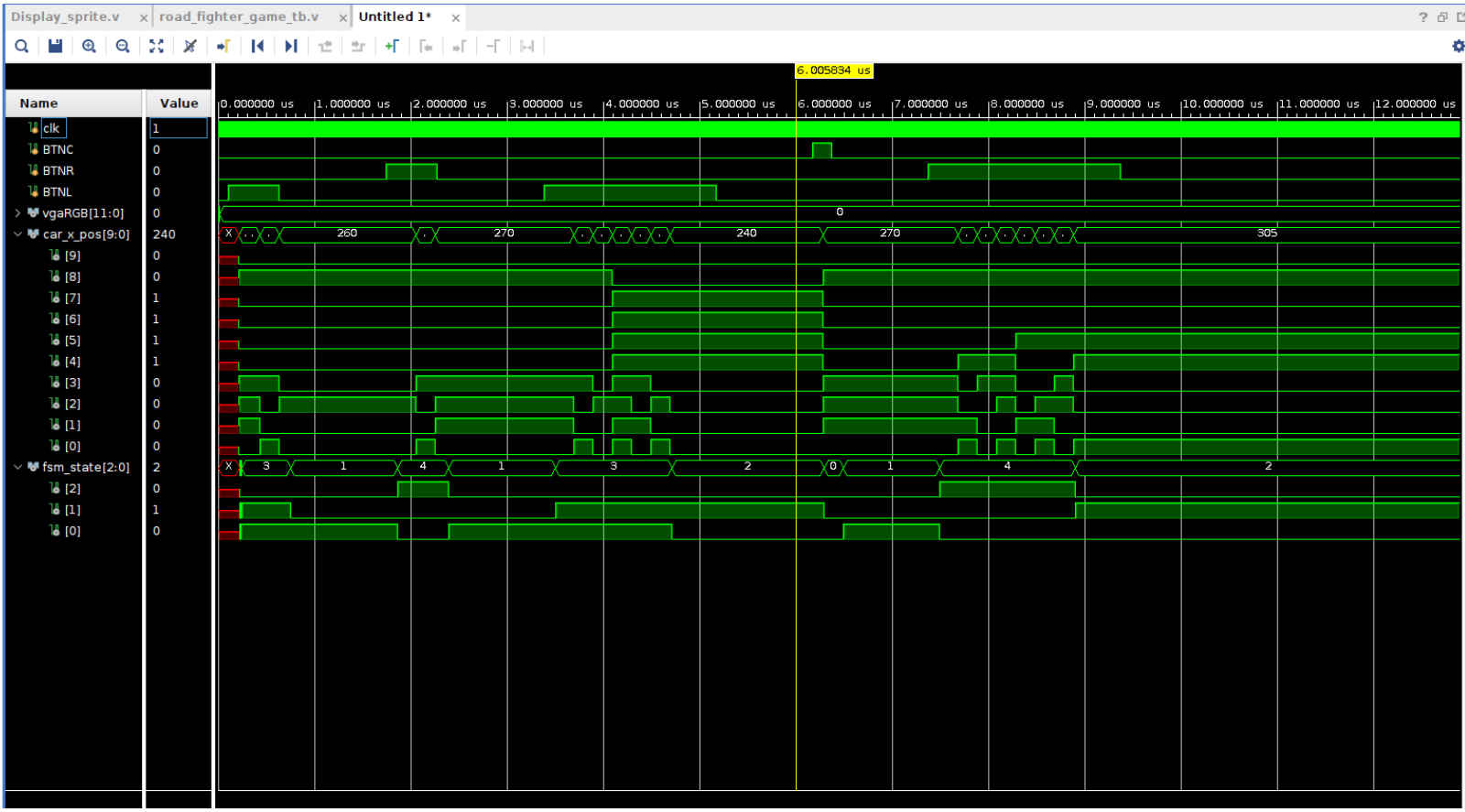

Figure 3 Shows all states of the movement and all collisions.

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.

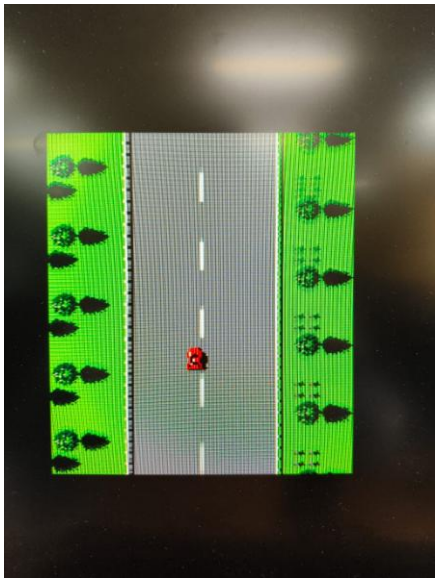

Figure 6 Start

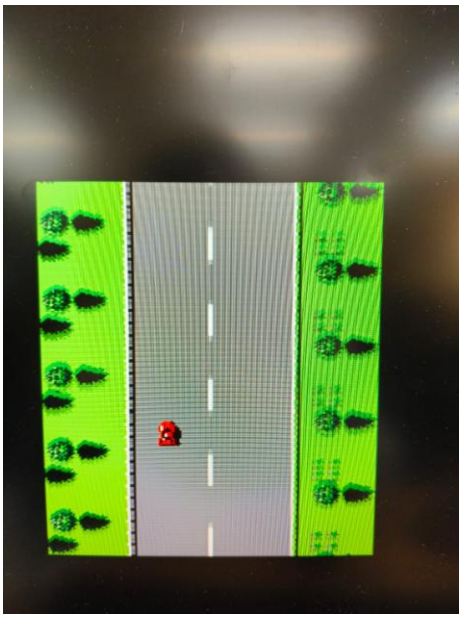

Figure 5 Left movement

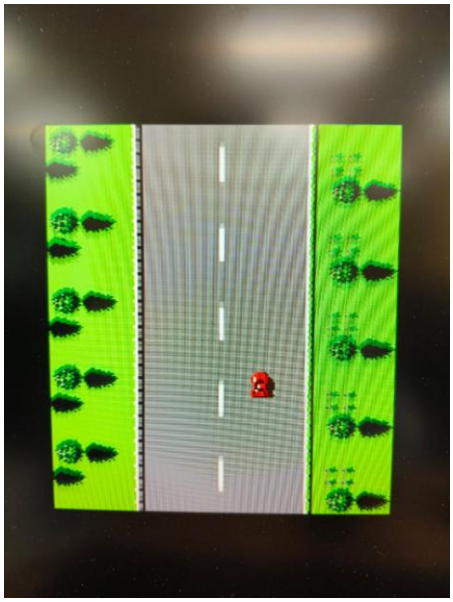

Figure 8 Right movement

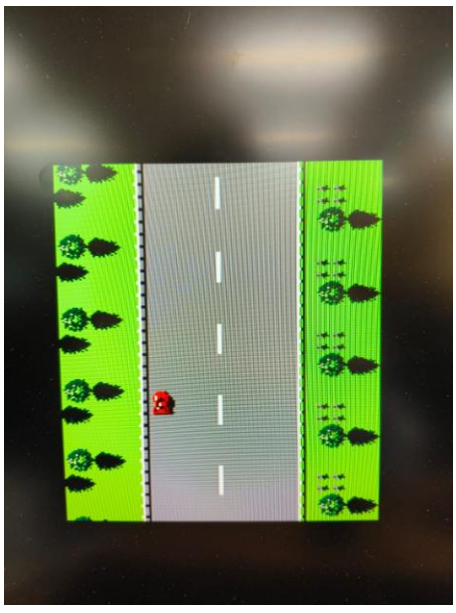

Figure 7 Left collision

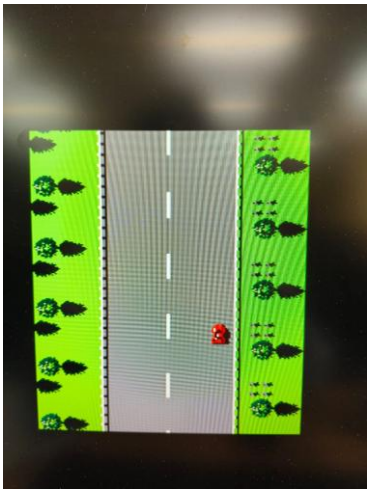

Figure 9 Right collision

5. Resource Utilization

Resource utilization after synthesis and implementation.

- No on-chip Block RAMs are consumed, meaning the memory footprint is extremely light and scalable.
- The design uses only a very small fraction of the available LUTs, showing that the combinational logic requirements are minimal.
- The low resource demand ensures easy timing closure and reliable performance on the FPGA.

Resource Type	Used	Available	Utilization (%)
LUTs	209	20800	1.00
Block RAMs	0	50	0.00
DSPs	0	90	0.00
Flip Flops	226	41600	0.54

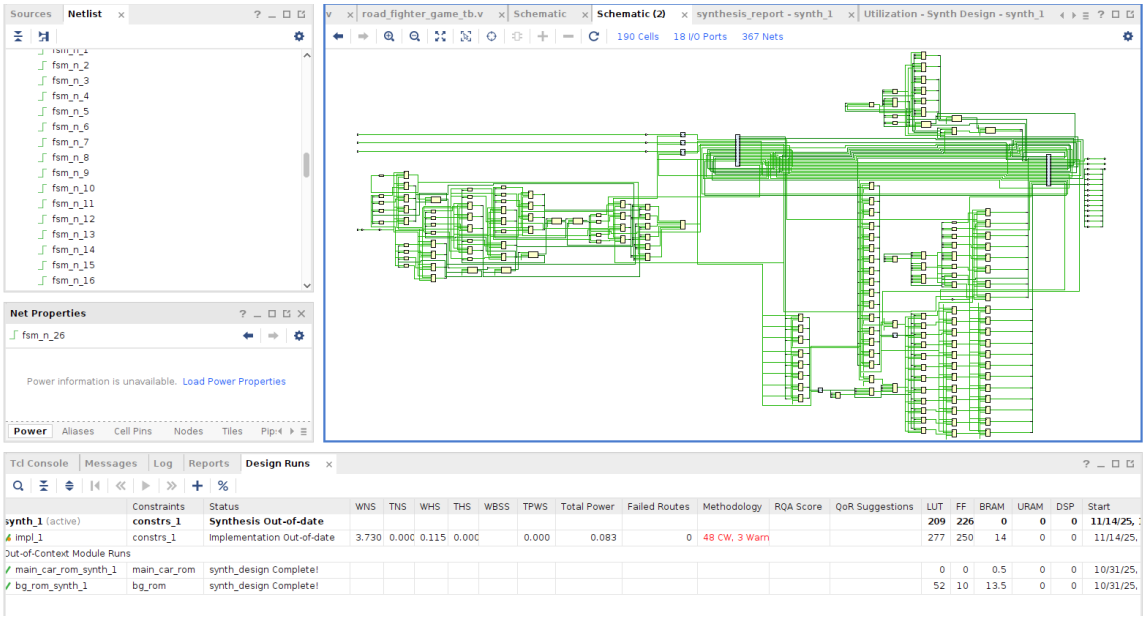

Figure 10 Luts and flip flops

6. Schematics

The schematic accurately represents the interconnection between the FSM, debouncers, VGA driver, 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

In this part, a finite state machine (FSM) was successfully designed and implemented to control the horizontal movement of the car and detect boundary collisions. The FSM efficiently managed transitions between movement, idle, and collision states, while debouncing ensured reliable button input. The car's movement and collision behavior were correctly displayed on the VGA screen, demonstrating smooth control and stable performance. This forms the foundation for further game logic and interactive elements in subsequent parts.