COL215 – Digital Logic and System Design

Department of Computer Science & Engineering, IIT Delhi Semester I, 2025–26

Lab Assignment 8 (Friday)

Developed By:
Bharmal Bhakar (2024CS10403)
Ritik Raj (2024CS10086)

1 Introduction

This report documents the implementation and results for Part I of the COL215 Lab Assignment 8: Road Fighter Game. The primary objective of Part I is to establish the fundamental video output infrastructure on the Basys 3 FPGA board using the Video Graphics Array (VGA) interface. This involves utilizing pre-designed modules (e.g., VGA_driver.v, counters) to achieve a standard 640 × 480 resolution and subsequently displaying a composite static image of the game background and the main car sprite. A critical task within this part is refining the display logic to correctly handle sprite transparency, ensuring the car is rendered seamlessly onto the road background by removing the dedicated pink color used as a transparent key

2. Design Decisions

The Display_sprite module was adapted from the provided template. It instantiates the VGA_driver and two memory modules (bg_rom and main_car_rom) to store the color data for the background and the car sprite, respectively.

1. clk_divider.v: Generates the 25MHz pixel_clock from the 100MHz system clock (clk) for VGA timing adherence (100MHz/4 = 25MHz).
2. Horizontal and Vertical Counters: (Horiz_counter.v and Vert_counter.v) Generate the current pixel coordinates (hor_pix, ver_pix) necessary for memory addressing and sync signal generation.
3. VGA_driver.v: Consolidates the clock and counter modules to generate the VGA synchronization signals (HS, VS) and the video_on signal, which indicates the active display area.
4. Display_sprite.v: The top-level rendering module for Part I. It uses the pixel coordinates to address two ROMs (Background and Car Sprite) and multiplexes their outputs to produce the final 12-bit VGA color signal (vgaRGB).

2.1. Sprite and Background Location Logic

The display logic determines whether the current pixel coordinate (hor_pix, ver_pix) falls within the bounds of the background or the car sprite.

Background Bounding Box:

- The background starts at an offset of (OFFSET_BG_X, OFFSET_BG_Y) = (200, 150).
- The logic checks if $200 \le$ hor_pix $< (200 + 160)$ and $150 \le$ ver_pix $< (150 + 240)$.

- If within bounds (bg_on = 1), the address for bg_rom is calculated as: bg_rom_addr ← (hor_pix - 200) + (ver_pix - 150) × 14.

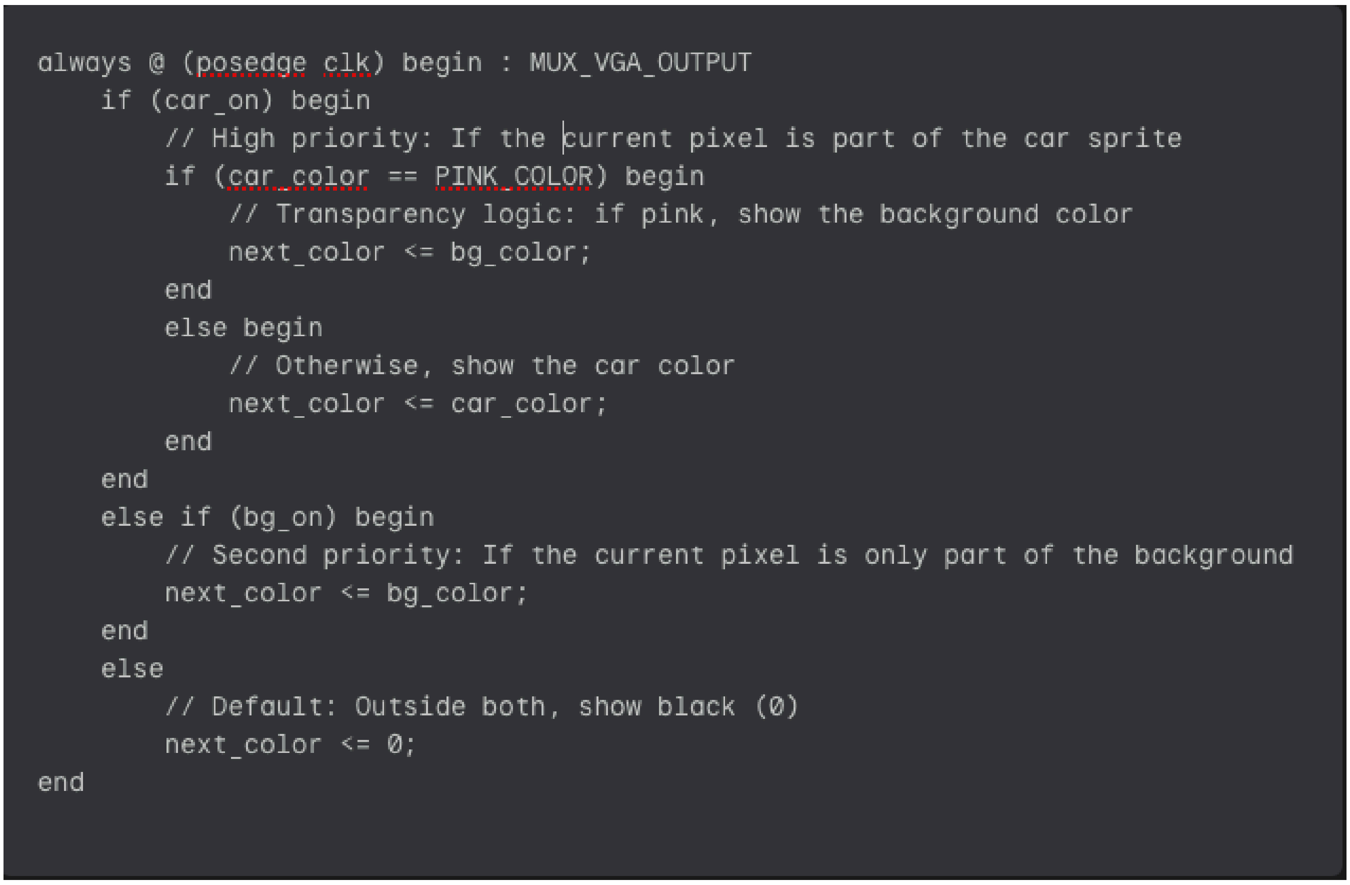
Car Sprite Bounding Box:

- The car is fixed at a starting position of (car_x, car_y) = (270, 300).
- The logic checks if $270 \le$ hor_pix < (270 + 14) and $300 \le$ ver_pix < (300 + 16).
- If within bounds (car_on = 1), the address for main_car_rom is calculated as: car_rom_addr ← (hor_pix - 270) + (ver_pix - 300) × 14.

2.2. Transparency Implementation

The assignment required implementing transparency for the car sprite by treating the pink color (12'b101000001010) as a "transparent" key. This logic is implemented in the MUX_VGA_OUTPUT always block.

Design Decision: The priority of the car sprite over the background is ensured by checking the car_on signal first.



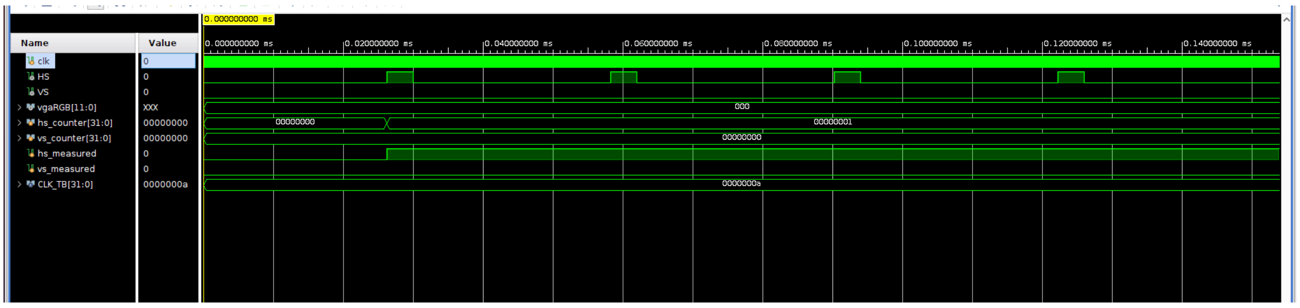
```
always @ (posedge clk) begin : MUX_VGA_OUTPUT
    if (car_on) begin
        // High priority: If the current pixel is part of the car sprite
        if (car_color == PINK_COLOR) begin
            // Transparency logic: if pink, show the background color
            next_color <= bg_color;
        end
        else begin
            // Otherwise, show the car color
            next_color <= car_color;
        end
    end
    else if (bg_on) begin
        // Second priority: If the current pixel is only part of the background
        next_color <= bg_color;
    end
    else
        // Default: Outside both, show black (0)
        next_color <= 0;
end
```

2.3. TestBench

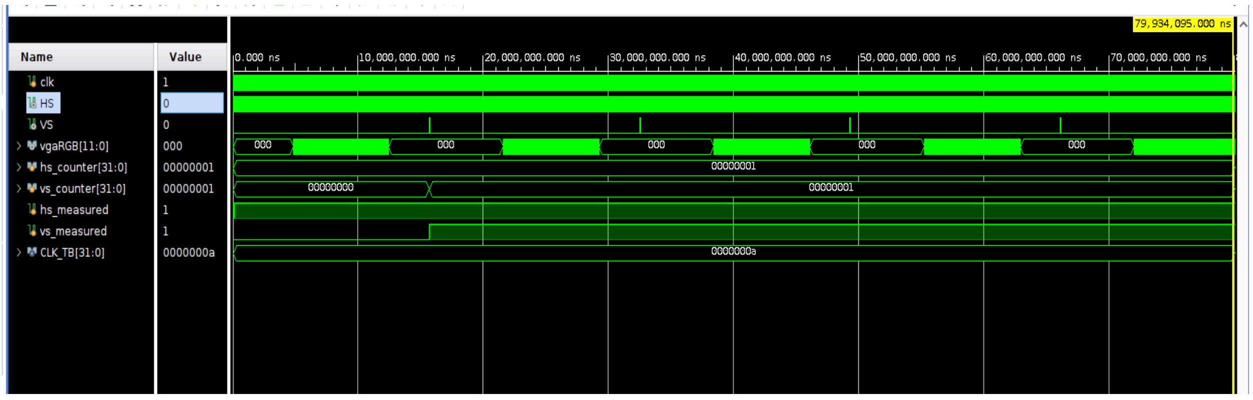
A dedicated test bench was created to verify the functionality of all modules. The primary focus of the simulation was to validate the VGA timing parameters, specifically checking the behavior of the ‘HS‘ (Horizontal Sync) and ‘VS‘ (Vertical Sync) signals and the generated pixel coordinates. The test bench was clocked using the 100MHz system clock, and the simulation was run long enough to observe at least two complete frames to confirm the stability of the VGA signals and the correct calculation of the pixel clock.

3. Simulation Snapshots

The module was simulated using a test bench to analyze the VGA synchronization signals (HS and VS). The measured values from the simulation, given a clock period of 10 ns per clock cycle, are recorded and analyzed below:



In this picture we can see the horizontal sync(HS), in which Gap between Successive Pulses is **3.2 K** (3200) clock cycles which is equals to 32000 ns which is time for one scanline.



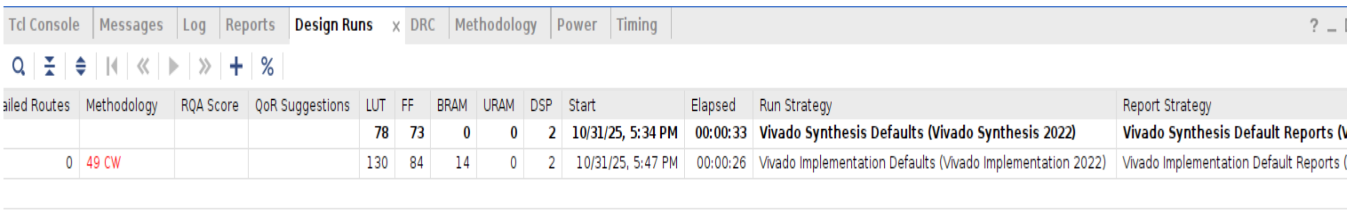
We can see the vertical sync(VS). Gap between successive pulses is **1.68 Million** (1,680,000) clock cycles which is equals to 16,800,000 ns which shows the time for one videoframe.

Timing Consistency Check: The relationship provided, 1 VS = 525 × 1HS, means one frame is composed of 525 scanlines. We can verify this relationship with the cycle counts:

$$\frac{VS\ Cycles}{HS\ Cycles} = \frac{1{,}680{,}000}{3{,}200} = 525$$

The measured cycle counts are perfectly consistent with the standard VGA timing resolution of 640 ×480(which uses 525 total lines per frame), confirming the correct functionality of the VGA_driver module.

4. Synthesis Report

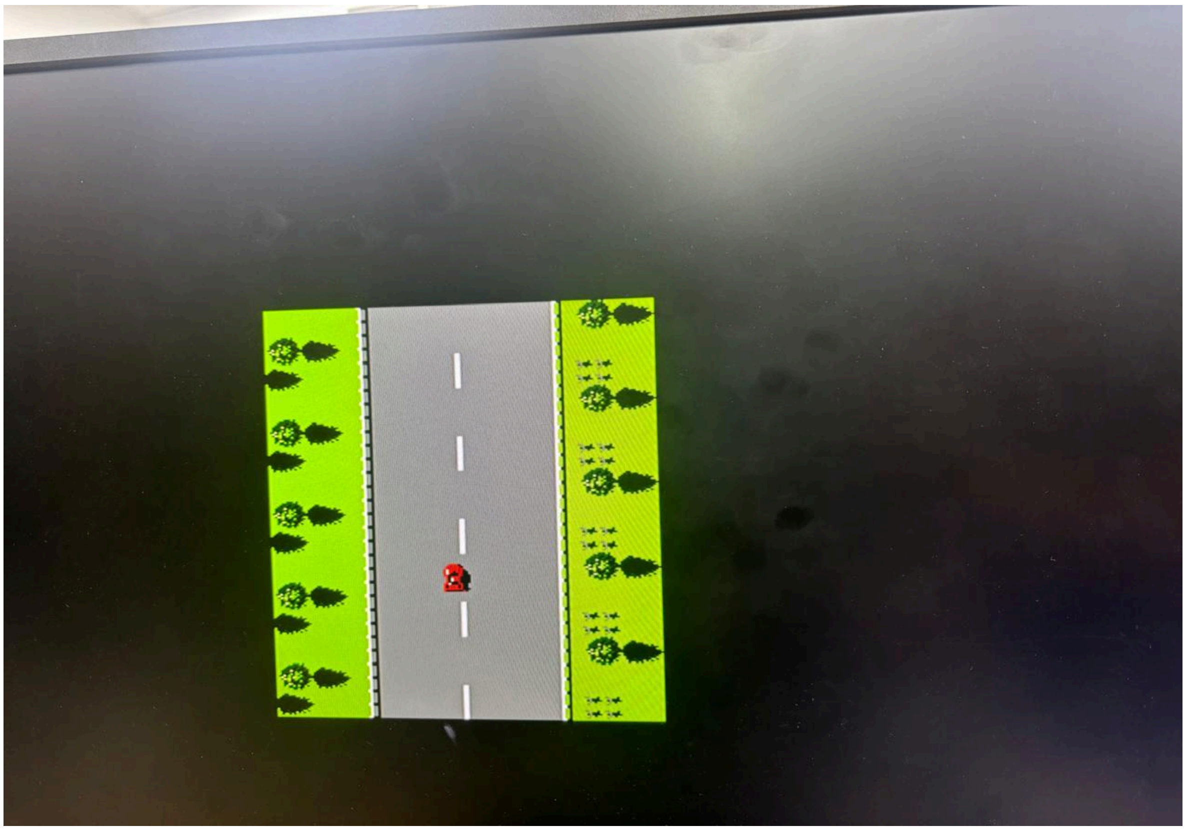


ailed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
				78	**73**	**0**	**0**	**2**	**10/31/25, 5:34 PM**	**00:00:33**	**Vivado Synthesis Defaults (Vivado Synthesis 2022)**	**Vivado Synthesis Default Reports (V**
0	49 CW			130	84	14	0	2	10/31/25, 5:47 PM	00:00:26	Vivado Implementation Defaults (Vivado Implementation 2022)	Vivado Implementation Default Reports (

The implementation uses 78 LUTs for all combinatorial logic, which includes the pixel boundary checks (for both background and car) and the critical 12-bit color multiplexer (MUX) that implements the transparency feature (comparing car_color to PINK_COLOR). The 73 Flip-Flops (FFs) are utilized for the synchronous sequential elements, such as the bg_on, car_on status flags, and the registered color pipeline (output_color) required for stable VGA output. Crucially, the utilization report shows 0 BRAMs and 0 DSPs used by the Display_sprite module itself. This confirms that the module is purely a control and logic path. The actual memory used for the background and car data is implemented via separate Block Memory Generator IP cores (which consume the FPGA's BRAM resources but are not attributed to this module's logic count), resulting in a highly efficient and resource-minimal design for image display.

5. On-Board Implementation

This implementation successfully displays the road background and the car sprite on the VGA, using a color multiplexer to achieve **sprite transparency** by replacing the car's pink background color with the corresponding background road color.

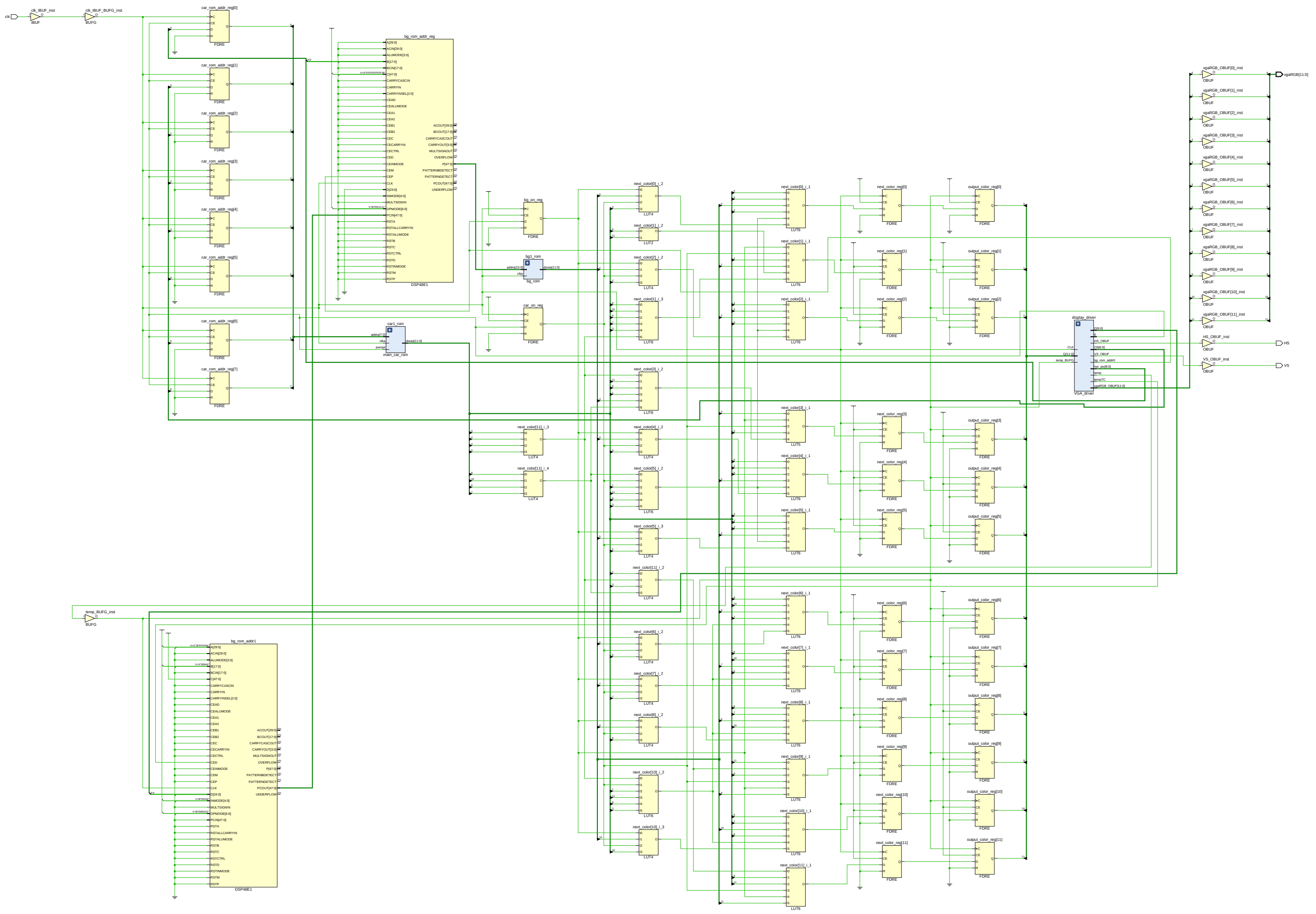


6. Conclusion

Part I successfully achieved the objective of displaying a stationary background and a car sprite with functional transparency on the VGA display. The core achievement was the implementation of a prioritized color MUX within the Display_sprite module that correctly overlays the car onto the road by treating the designated pink color as a transparent key. Simulation results verified the standard VGA timing relationship 1VS = 525 ×1HS with a line-time of 32.0 μs and a frame-time of 16.8 ms.

7. Generated Schematic

The Generated Schematic for part1 of this project is attached below


clk
clk_IBUF_inst
IBUF
clk_IBUF_BUFG_inst
BUFG
car_rom_addr_reg[0]
car_rom_addr_reg[1]
car_rom_addr_reg[2]
car_rom_addr_reg[3]
car_rom_addr_reg[4]
car_rom_addr_reg[5]
car_rom_addr_reg[6]
car_rom_addr_reg[7]
FDRE
bg_rom_addr_reg
DSP48E1
bg_on_reg
bg1_rom
bg_rom
car_on_reg
car1_rom
main_car_rom
next_color[11]_i_3
next_color[11]_i_4
LUT4
LUT6
display_driver
VGA_driver
vgaRGB_OBUF[0]_inst
vgaRGB_OBUF[11]_inst
OBUF
vgaRGB[11:0]
HS_OBUF_inst
HS
VS_OBUF_inst
VS
temp_BUFG_inst
bg_rom_addr1
next_color_reg[0]
next_color_reg[11]
output_color_reg[0]
output_color_reg[11]