# Lab 3: Clock Divider and LED Controller

Submission Due Dates:

```
Demo:            2022/10/04 17:20
Source Code:     2022/10/04 18:30
Report:          2022/10/09 23:59
```

## Objective

1. Getting familiar with the clock divider and LED control on the FPGA demo board.
2. Practicing finite-state machines (FSMs) in Verilog.

## Action Items

### 1 `Module: clock_divider` (10%)

Write a Verilog module for the clock divider that divides the frequency of the input clock by $2^{25}$ to get the output clock. Here is the template you should use:

```
module clock_divider #(parameter n=25) (
  input clk,
  output clk_div
);

   // add your design here

endmodule
```

### 2 `lab3_1.v` (30%)

Write a Verilog module of the LED controller, which is synchronous with the positive clock edges. The clock frequency is obtained by dividing Basys3's onboard 100MHz clock by either $2^{24}$ or $2^{27}$. Also, download and run your LED controller on the FPGA board. You must use this clock divider in the following designs (e.g., lab3_2.v). Note that you may design with or without finite-state machines (FSMs).

**Here are the specification of inputs and outputs:**

- ✓ If **rst**: Set only LD15 to ON.
  - The reset is asynchronous and positive edge triggered.
- ✓ If **en** == **0**: hold the status of LEDs unchanged.
- ✓ If **en** == **1**: The ON-ed LED will go from left to right. When it hits the end, it will start from LD15 again.
  - If **speed** == **1**: The ON-ed LED will move at the clock rate of (100 MHz / $2^{27}$).
  - If **speed** == **0**: The ON-ed LED will move at the clock rate of (100 MHz / $2^{24}$).

**Example: Upon reset**

(LD15) ●○○○○○○○○○○○○○○○ (LD0) (●: LED on, ○: LED off)

**IO Connection:**

| | |
|---|---|
| clk | connected to W5 |
| rst | connected to W17 |
| en | connected to V17 |
| speed | connected to V16 |
| led | connected to LD15-LD0 |

You should use the following template for your design and name the file as “lab3_1.v”.

```
module lab3_1(
   input clk,
   input rst,
   input en,
   input speed,
   output [15:0] led
);

   // add your design here
   // output signals can be reg or wire


endmodule
```

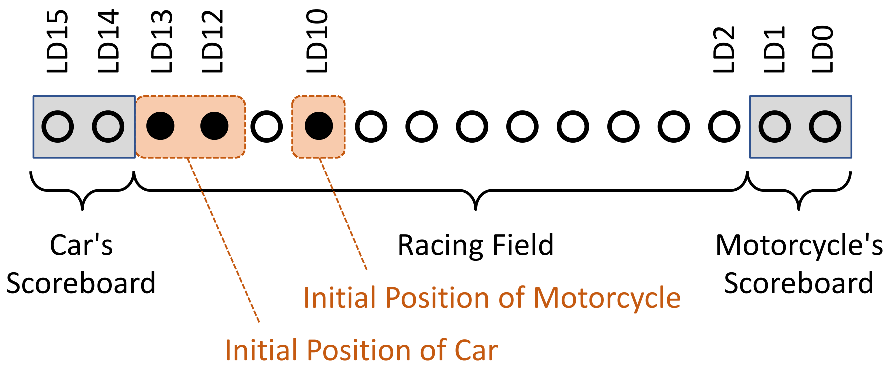
**Demo:** https://www.youtube.com/watch?v=-_9kpBulsCk

## 3 `lab3_2.v` (60%)

Batman is developing his new Batmobile, a heavily armored car, and a motorcycle for Robin. To verify both vehicles, he needs to emulate their functionality first. Your job is to help the superhero build up this emulator and, of course, save the world.

Write a Verilog module of the LED controller (the racing emulator), which is synchronous with the positive clock edges. This controller will perform the racing of Batman's Car and Robin's Motorcycle. The vehicles operate at two different rates: the fast rate is (100 MHz / $2^{24}$); the normal rate is (100 MHz / $2^{27}$).

As shown in the following figure, the LD15 and LD14 are reserved for the scoreboard of the Car. LD1 and LD0 are reserved for the scoreboard of the Motorcycle. After reset, the Car will start from LD12 and LD13 (it occupies two LEDs); the Motorcycle will start from LD10. Once en == 1, both vehicles will race from left to right at the normal rate. Their goal is LD2. The first vehicle to reach the goal wins the round. The winner's scoreboard will increase by one. And the loser should stop its action. Later, the controller will reset the racing field. The racing will finish when any vehicle reaches four points (i.e., anyone wins four rounds). Your controller should enter the finish state for one cycle of (100 MHz / $2^{27}$) with all the LEDs in the racing field turned on. The controller then goes to the initial state.



In addition, the Car comes with a turbo mode (with the **speed switch**): during each round, it will have a chance to speed up for five consecutive cycles. To prevent overheating, you should ensure the Car won't enter the turbo mode twice in a round. It also equips a weapon to freeze the Motorcycle when the **freeze switch** is pulled up.

In lab3_2, you must design with **finite state machines** to perform the following six steps:

- Initialization
- Racing
- The Motorcycle finishes a round
- The Motorcycle wins the competition.
- The Car finishes a round
- The Car wins the competition.

**Hint:**

There are two alternatives to FSM clocking.

1. **Method 1:**
   The FSM can operate at the fast clock rate of (100 MHz / $2^{24}$) and emulate the Car at the fast speed. In addition, counters can be used to slow down the operating to the normal speed.
2. **Method 2:**
   You can generate two clock signals: the fast rate of (100 MHz / $2^{24}$) and the normal rate of (100 MHz / $2^{27}$). The FSM can operate at the normal clock rate since both vehicles race at the normal clock initially. Then, you can use a selector to switch between the two clock signals to control the Car's movement.
   (You probably may feel that this method is more intuitive in this lab. But there are pros and cons. Either is acceptable to accomplish this lab.)

**Further Descriptions:**

1. **Initialization:**
   You should reset all variables.

2. **Racing:**
   The LED display should be controlled at the fast clock rate. You may have two variables: one is to record the Car's current location, and the other is to record the Motorcycle's current location at the normal clock rate.

   When the **freeze switch** is pulled up, the Motorcycle's location counter will stop counting and remain at its current value until it is pulled down.

   When the **speed switch** is pulled up, the next five positions of the Car should be changed at the fast rate, and then the operating returns to the normal rate. You are required to prevent speeding up twice in a round.

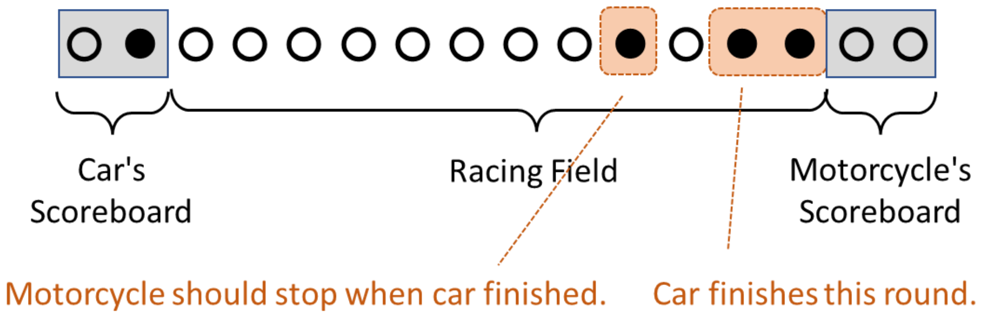
   Meanwhile, your state machine should keep checking the Motorcycle's current location and the Car's current location. Whichever reaches LD2, the state machine should stop counting the positions and enter the corresponding state. If both vehicles reach LD2 simultaneously, the Motorcycle wins.

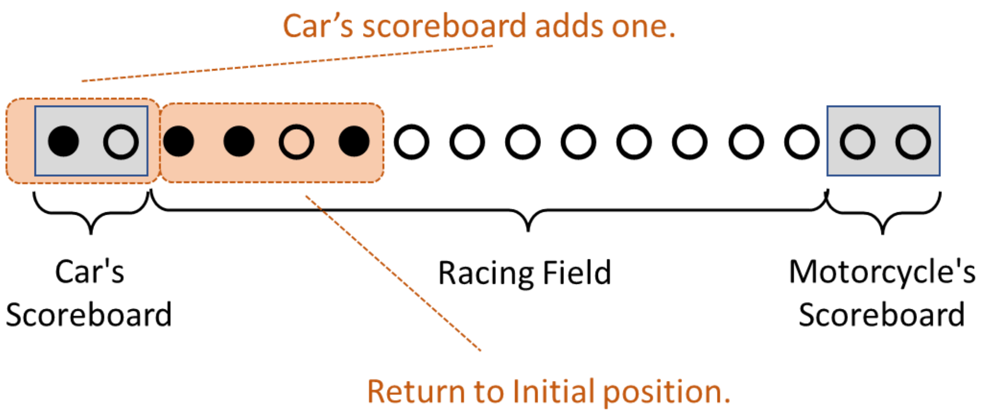
3. **The Car or Motorcycle finishes a round:**
   Whichever finishes a round, you should add its scoreboard by one and reset both locations of the Car and Motorcycle. The next round will begin after a normal cycle (100 MHz / $2^{27}$).

4. **The Car or Motorcycle wins the competition:**
   Whichever wins the competition, you should display the corresponding LED pattern. If the Car wins the competition, LD15 to LD2 should light on, and LD1 to LD0 should turn off. If the Motorcycle wins the competition, LD15 to LD14 should light off, and LD13 to LD0 should turn on. Then the FSM returns to the initialization state for the next competition. The next competition will begin after a normal cycle (100 MHz / $2^{27}$).

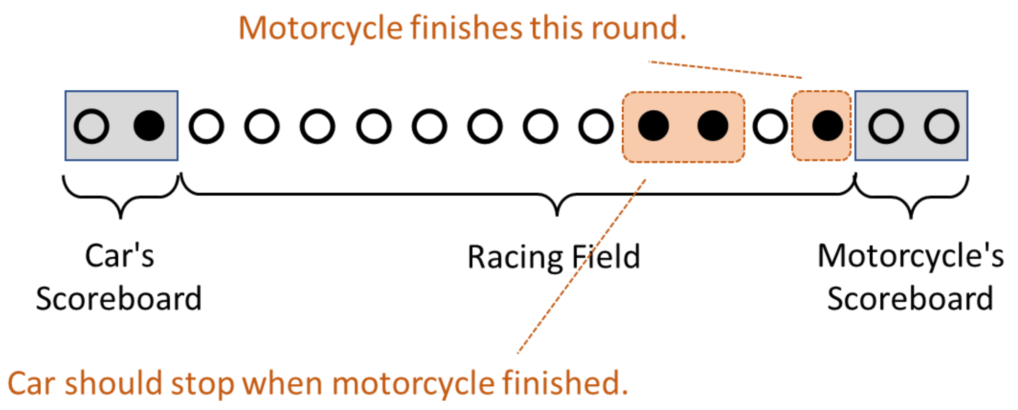
**Several scenarios are listed as follows for your reference:**

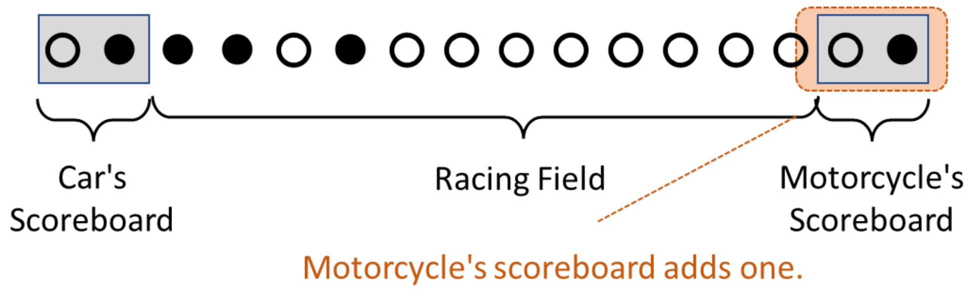
1. When the Car finishes one round with one point:
   Both vehicles reset to their initial positions; the Car gets two points.



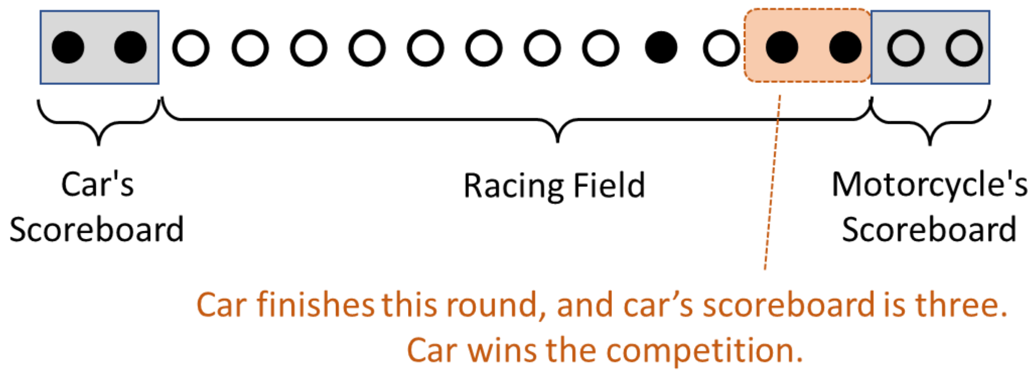


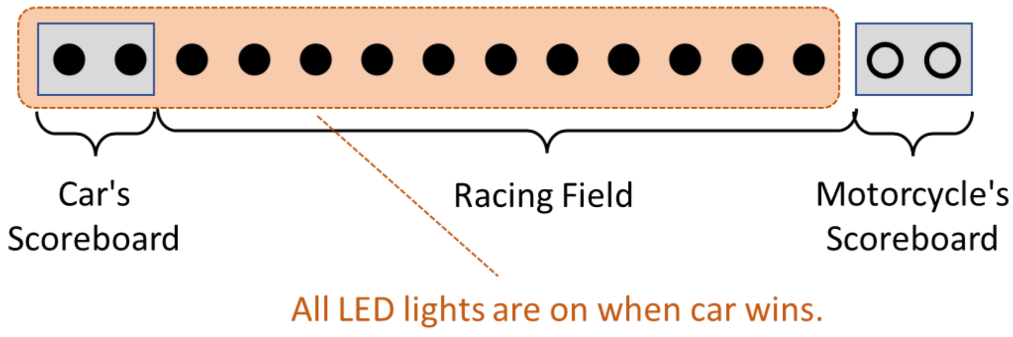
2. When the Motorcycle finishes one round with zero points:
   Both vehicles reset to their initial positions; the Motorcycle gets one point.


Motorcycle finishes this round.
Car's Scoreboard
Racing Field
Motorcycle's Scoreboard
Car should stop when motorcycle finished.

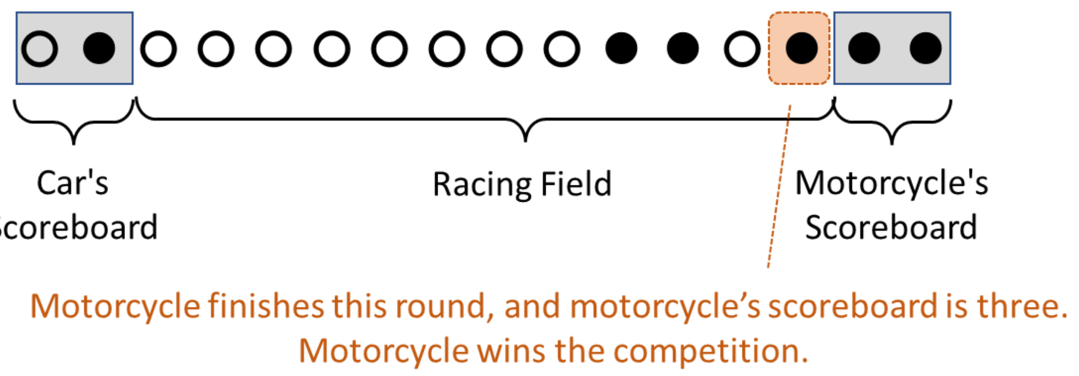


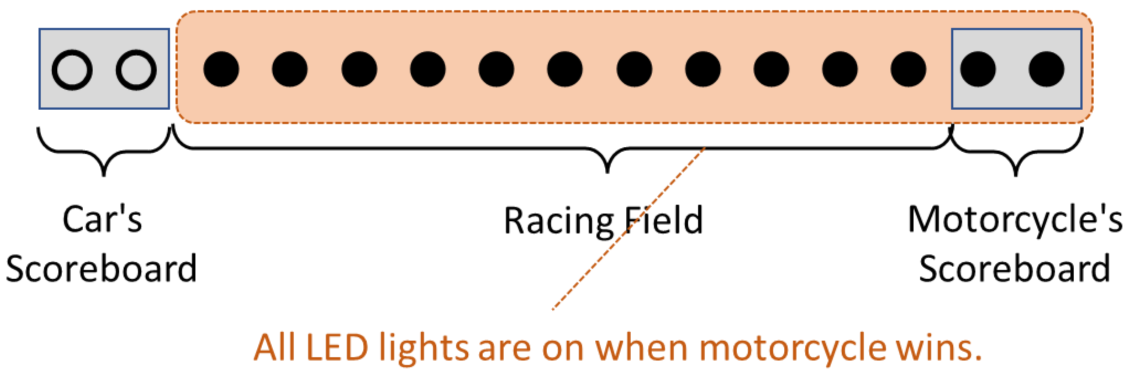
3. When the Car wins four rounds:
   The controller enters the finish state with all the LEDs in the racing field turned on.


Car's Scoreboard
Racing Field
Motorcycle's Scoreboard
Car finishes this round, and car's scoreboard is three.
Car wins the competition.

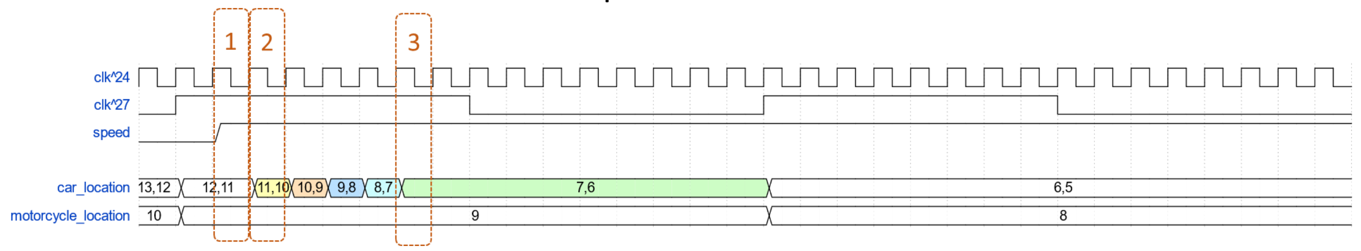


4. When the Motorcycle wins four rounds:
   The controller enters the finish state with all the LEDs in the racing field turned on.


Car's Scoreboard
Racing Field
Motorcycle's Scoreboard
Motorcycle finishes this round, and motorcycle's scoreboard is three.
Motorcycle wins the competition.

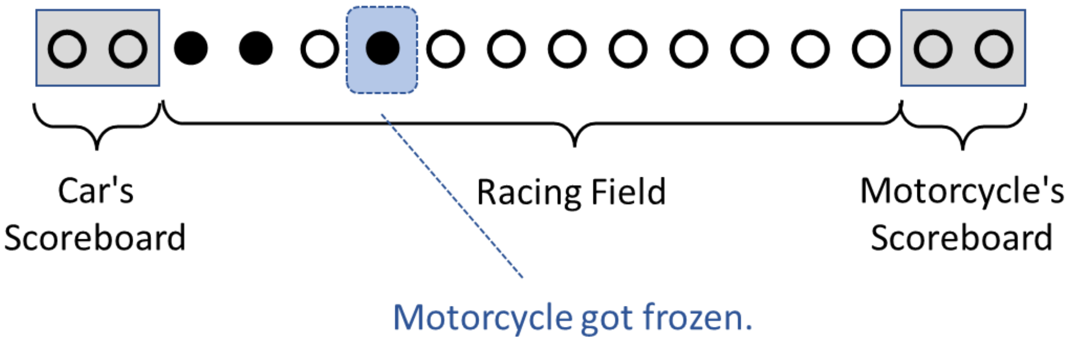


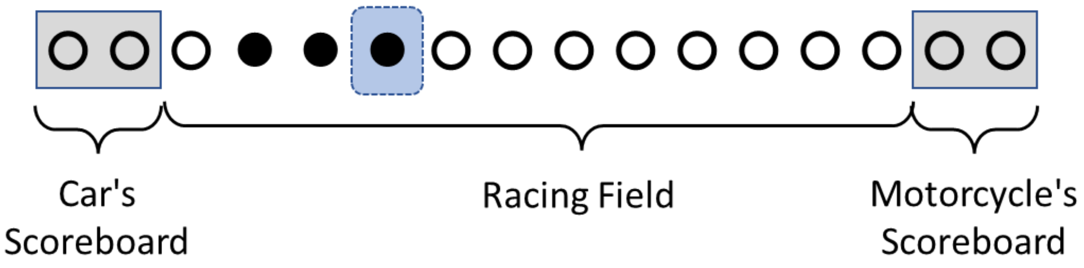
5. About the turbo mode with the speed switch:

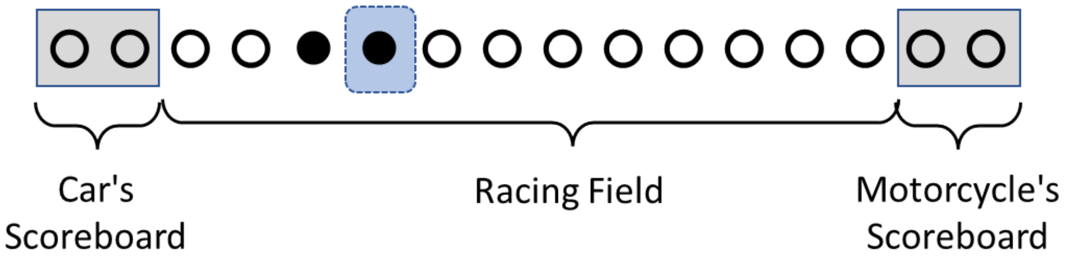


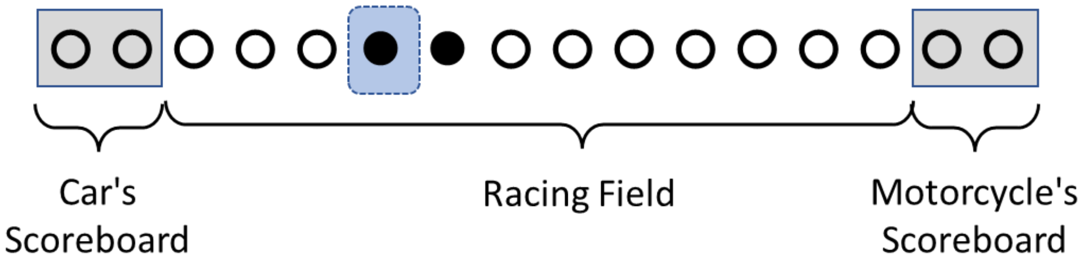
   (1) The speed switch is pulled up.
   (2) The car location (`car_location` in the figure) is in sync with the fast clock (`clk^24` in the figure) for five cycles.
   Note: The `car_location` and `motorcycle_location` are for illustration purposes only, and you may use different representations to record the vehicle locations.
   (3) After the five cycles, the car location (`car_location`) is in sync with the normal clock (`clk^27`) again.

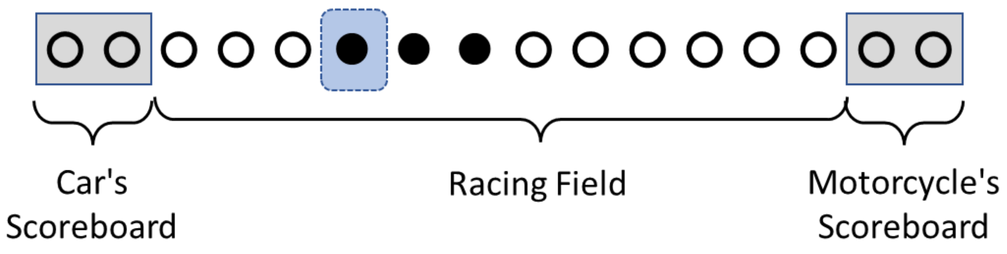
6. When the Motorcycle got frozen:

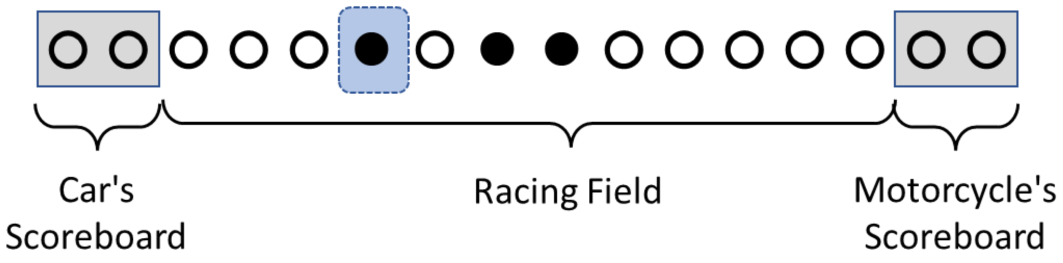


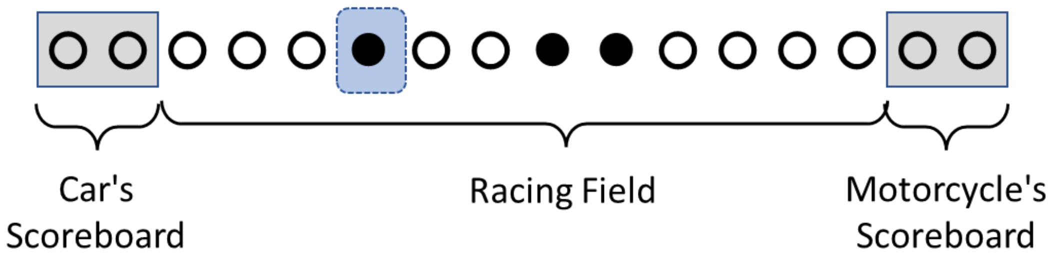












Scoreboard counting

○○ : 0

○● : 1

●○ : 2

●● : 3

**Here are the specification of inputs and outputs:**

✓ If **rst**: reset to initial state.
  - The reset is positive edge triggered.

✓ If **en** == **0**: hold all LEDs unchanged.

✓ If **en** == **1**: Start the racing at the normal clock rate.
  - If **speed** == **1**: the Car will move at the fast rate of (100 MHz / $2^{24}$) for five consecutive cycles and change back to the normal rate of (100 MHz / $2^{27}$) for the rest of the round. In the next round, the Car will be able to speed up again.
  - If **speed** == **0**: Both vehicles will move at the normal rate.
  - If **freeze == 1**: The Motorcycle will freeze (i.e., stay at its current location).

- If **freeze == 0**: The Motorcycle moves as normal.

**IO Connection:**

| | |
|---|---|
| clk | connected to W5 |
| rst | connected to W17 |
| en | connected to V17 |
| speed | connected to V16 |
| freeze | connected to W16 |
| led | connected to LD15-LD0 |

You should use the following template for your design and name the file as "lab3_2.v".

```
module lab3_2(
   input clk,
   input rst,
   input en,
   input speed,
   input freeze,
   output [15:0] led
);

   // add your design here
   // output signals can be reg or wire


endmodule
```

**Demo:** https://www.youtube.com/watch?v=zNbnmzzecRw

## 4 Questions and Discussion

Please answer the following questions in your report.

A. Robin feels that the freezing weapon in Batmobile is not fair. He upgrades his motorcycle with a shield such that the Motorcycle can only be frozen for three consecutive cycles in a round. How do you modify your racing emulator design?

B. In lab3_2, if you implemented Method 1, discuss how you can implement Method 2. Or if you implemented Method 2, discuss how you can implement Method 1. Discuss your concept as detailed as possible, and you are encouraged to use block diagrams and FSMs to illustrate your design.

## 5 Guidelines for the report

Your report should include but not limit to the following items.

A. **Lab Implementation (35%)**

You may elaborate on the followings.

1. Block diagram of the design with explanation
2. Partial code screenshot with the explanation: you don't need to paste the entire code into the report. Just explain the kernel part.
3. Event-based FSM with the explanation:
   Note1: Use events to describe the transition of FSM but not signals logic statement.
   Note2: You don't need to do this part if there is no FSM in this lab.

B. **Questions and Discussions (50%)**

Provide your answer to the Questions and Discussions in the lab assignment.

C. **Problem Encountered (10%)**

Describe the problems you encountered, solutions you developed, and the discussion. Explaining them with code segments or diagrams is recommended.

D. **Suggestions (5%)**

Any suggestions for this course are more welcome.
(If not, you may also post a joke. A funny one, please. It is not mandatory and has nothing to do with the grading. But it would undoubtedly amuse us. ☺)

## Attention

- ✓ Please refer to Lab 0 and Lecture 03 for the FPGA implementation flow using Vivado.
- ✓ You may need to change your runtime (ns) in "Simulation Settings" to fit the testbench settings before the simulation.
- ✓ You should hand in **lab3_1.v, and lab3_2.v**. If you have multiple modules for an Action Item, you must integrate them into a single Verilog file (e.g., lab3_1 and clock_divider in lab3_1.v; lab3_2 and clock_divider in lab3_2.v, etc.).
  - ■ **Upload each source file directly, DO NOT hand in a compressed ZIP or RAR file!**
- ✓ **DO NOT** copy-and-paste code segments from the PDF materials. Occasionally, it will also paste invisible non-ASCII characters, leading to hard-to-debug syntax errors.
- ✓ You should also hand in your report as **lab3_report_StudentID.pdf** (i.e., lab3_report_108080001.pdf).
- ✓ You should be able to answer questions of this lab from TA during the demo.
- ✓ You need to generate the bitstream files before the lab demo. Prepare separate bitstream files for lab3_1, and lab3_2, respectively, to make the demo process smooth.