REDTAIL LABORATORY EXERCISE

Directional Wiper Controller

DE1-SoC with VHDL

LEVEL **Intermediate** | TIME **90-120 minutes** | PLATFORM **Altera DE1-SoC**

Design and test a stateful two-bit controller for a simulated vehicle windshield wiper. Unlike the one-bit Wiper, this simulation expects your VHDL design to remember the travel direction and reverse it at each endpoint.

The Wiper 2-Bit 3D simulation and original remote-laboratory exercise were created by Giovanna Lani and WebLab-Deusto at the University of Deusto in Spain. This lesson adapts their directional VHDL challenge to the current REDTAIL mapping. The completed implementation is provided separately to verified instructors.

Figure 1. Wiper 2-Bit 3D simulation by Giovanna Lani and WebLab-Deusto, University of Deusto (Spain).

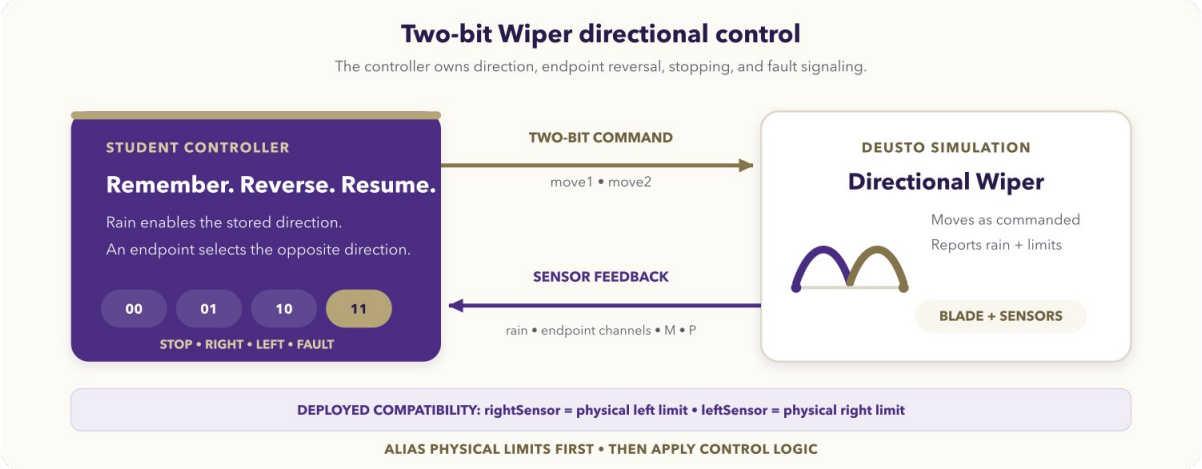

Figure 2. Directional two-bit Wiper control loop.

Learning objectives

After completing the activity, you should be able to:

- design a small synchronous state machine in VHDL;
- encode stop, right, left, and fault commands on two outputs;
- preserve direction through a dry pause and resume;
- reverse immediately and safely at physical travel limits;
- isolate runtime signal names from their physical meanings with aliases;
- verify reset, endpoint, persistence, and contradictory-sensor behavior.

Prerequisites

You should know VHDL entities, enumerated state types, clocked processes, combinational output logic, and `std_logic`. You need the DE1-SoC VHDL Code-IDE and access to the Wiper 2-Bit simulation.

Open the REDTAIL Wiper 2-Bit simulation and the current DE1-SoC mapping before starting.

Compatibility contract

The deployed two-bit simulation uses endpoint signal names that are reversed relative to their physical locations:

Runtime input	Physical meaning in this simulation
`rightSensor`	Blade is at the **physical left-hand limit**.
`leftSensor`	Blade is at the **physical right-hand limit**.

Create semantic aliases such as `at_left_limit` and `at_right_limit` as soon as you read the inputs. Use those aliases throughout the state machine. This compatibility contract is intentional for this exercise and must not be "corrected" by changing pins or renaming the deployed channels.

Required behavior

The output pair is interpreted in `(move1, move2)` order:

move1	move2	Physical command
0	0	Stop
0	1	Move physically right
1	0	Move physically left
1	1	Fault

Your core controller must:

1. stop while rain is inactive;
2. begin by moving physically right after reset when rain is active and no endpoint supplies a direction;
3. reverse toward the left at the physical right endpoint;
4. reverse toward the right at the physical left endpoint;
5. remember the most recent direction during a dry pause and resume it;
6. output `11` whenever both endpoint sensors are high;
7. ignore M and P in the required solution.

DE1-SoC signal mapping

Runtime signal	VHDL access	Direction	Semantic use
M button	`V_GPIO(23)`	Simulation to FPGA	Optional extension
P button	`V_GPIO(24)`	Simulation to FPGA	Optional extension
`move1`	`V_GPIO(26)`	FPGA to simulation	First command bit
`move2`	`V_GPIO(27)`	FPGA to simulation	Second command bit
Rain sensor	`V_GPIO(28)`	Simulation to FPGA	Movement request
`rightSensor`	`V_GPIO(29)`	Simulation to FPGA	**Physical left limit**
`leftSensor`	`V_GPIO(30)`	Simulation to FPGA	**Physical right limit**
Reset	`V_BT(0)`	Board to design	Active low

The virtual GPIO indices are LabsLand bus positions. The REDTAIL mapping is authoritative over tables in older development versions of the Deusto guide.

Assignment

1. Model direction explicitly

Draw a two-state diagram with `toward_right` and `toward_left`.. Add transitions for both physical endpoints and show which conditions preserve state. Treat rain as an output enable, not as a reason to erase the remembered direction.

2. Create the VHDL top level

Create `wiper_2bit_deusto.vhd`:

```
library ieee;
use ieee.std_logic_1164.all;
entity wiper_2bit_deusto is
    port (
        CLOCK_50 : in    std_logic;
        V_BT     : in    std_logic_vector(3 downto 0);
        V_GPIO   : inout std_logic_vector(35 downto 23);
        G_LED    : out   std_logic_vector(9 downto 0) := (others => '0')
    );
end entity wiper_2bit_deusto;
```

Complete the architecture yourself. Define physical endpoint aliases, update direction in a clocked process, and derive both command bits in complete combinational logic. Drive only `V_GPIO(26)` and `V_GPIO(27)`.

3. Review priority and timing

Give the dual-endpoint fault highest output priority. Make an endpoint reversal effective immediately in the command logic so the controller cannot continue driving into an active physical limit while waiting for the next clock edge. Explain how the direction register makes a dry resume deterministic.

4. Compile and program

1. Select the DE1-SoC VHDL environment and Wiper 2-Bit simulation.
2. Add the source and select `wiper_2bit_deusto` as the top level.
3. Synthesize and resolve relevant multiple-driver, latch, or undriven warnings.
4. Program the FPGA and observe physical travel direction beside the command bits.

Required test sequence

Test	Action	Expected command and behavior
Reset, dry	Hold reset, then release it with rain off.	`00`; default stored direction is right.
Rain start	Enable rain between endpoints.	`01`; blade moves physically right.
Right endpoint	Activate physical right limit (`leftSensor`).	`10`; blade reverses physically left.
Dry pause	Disable rain while travelling left.	`00`; left direction is retained.
Dry resume	Re-enable rain away from the endpoints.	`10`; left travel resumes.
Left endpoint	Activate physical left limit (`rightSensor`).	`01`; blade reverses physically right.
Button isolation	Press M and P while dry.	`00`; neither button changes core behavior.
Sensor fault	Activate both endpoint inputs.	`11`; no movement command is issued.
Fault recovery	Clear the contradiction with rain active.	Travel resumes according to the coherent endpoint or stored direction.

Deliverables

Submit the state diagram, compatibility aliases, `wiper_2bit_deusto.vhd`, test evidence for every row, and a short explanation of dry direction persistence and fault priority.

Optional extensions

- Give M or P a documented behavior without changing the core rain policy.
- Synchronize simulation inputs before clocked use.
- Latch a visible diagnostic fault until acknowledgement.
- Add assertions for command validity, endpoint reversal, and dry-state retention.

Completion checklist

- Endpoint runtime names are translated to physical aliases exactly once.
- `00`, `01`, `10`, and `11` match the deployed command contract.
- Direction reverses at each physical endpoint and survives a dry pause.
- Dual endpoints produce `11`; M and P do not affect the core.
- Reset and all required live sequences have been demonstrated.