REDTAIL LABORATORY EXERCISE

Directional Wiper Controller

STM32 NUCLEO-WB55RG with Mbed OS

LEVEL **Intermediate** | TIME **90-120 minutes** | PLATFORM **STM32 NUCLEO-WB55RG**

Implement a stateful two-bit windshield-wiper controller for the STM32 NUCLEO-WB55RG. Your Mbed application owns stop, physical direction, and the fault command; the simulation owns the rendered blade motion.

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 challenge to the current REDTAIL STM32 mapping. The completed application is available separately to verified instructors.

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

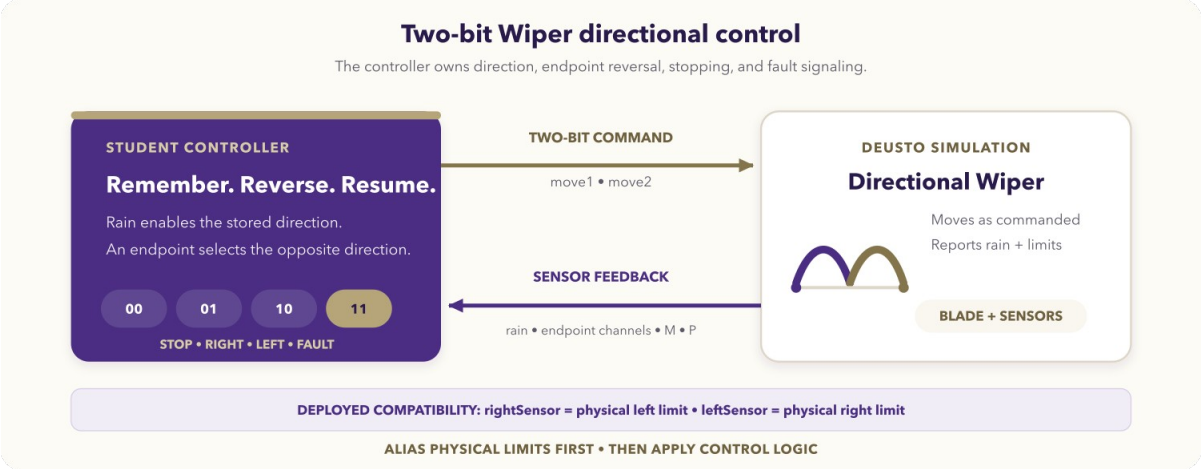

Figure 2. Directional two-bit Wiper control loop.

Learning objectives

After completing the activity, you should be able to:

- separate a stateful control policy from Mbed GPIO glue;
- represent stop, right, left, and fault as a two-bit command;
- retain travel direction across polling iterations and dry pauses;
- reverse at coherent physical endpoint feedback;
- contain a deployed naming mismatch with semantic aliases;
- host-test reset, sequence, fault, and optional-input isolation.

Prerequisites

You should know C++ functions, small structs or enums, Mbed `DigitalIn` and `DigitalOut`, and polling loops. You need Mbed OS 6 or Mbed OS CE for `NUCLEO_WB55RG` and access to the Wiper 2-Bit simulation.

Open the REDTAIL Wiper 2-Bit simulation and the current STM32 mapping before starting.

Compatibility contract

The deployed endpoint inputs are named for the opposite physical side:

Runtime input	Physical meaning
`rightSensor` / `PA_6`	Blade is at the **physical left-hand limit**.
`leftSensor` / `PB_9`	Blade is at the **physical right-hand limit**.

Read each pin once per loop and immediately create `at_left_limit` and `at_right_limit` values. Keep all policy code in physical terms. Do not swap the deployed pins.

Required behavior

The command pair is `(move1, move2)`:

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

The controller starts with rightward direction, moves only while rain is active, reverses at each physical endpoint, preserves direction across dry pauses, outputs `11` for simultaneous endpoints, and ignores M/P in the core.

STM32 signal mapping

Runtime signal	Mbed pin	Direction	Semantic use
Rain sensor	`PC_13`	Input	Movement request
`rightSensor`	`PA_6`	Input	**Physical left limit**
`leftSensor`	`PB_9`	Input	**Physical right limit**
M button	`PB_8`	Input	Optional extension
P button	`PC_12`	Input	Optional extension
`move1`	`PC_4`	Output	First command bit
`move2`	`PD_0`	Output	Second command bit

Initialize both outputs low before the polling loop. The current REDTAIL mapping is authoritative over tables in older Deusto development documents.

Assignment

1. Define a host-testable policy

Represent direction as an enum and the output as a two-Boolean command. Write a policy step that receives the previous direction plus one input snapshot and returns the next direction and current command. Keep Mbed types out of it.

2. Create the Mbed application

Create `main.cpp` and a small policy header:

```
enum class Direction { toward_right, toward_left };

struct Command {
    bool move1;
    bool move2;
};

// Define a policy state and a step function without GPIO access.

int main() {
    // Configure five inputs and two outputs, initially low.
    // Read one complete snapshot per loop.
    // Alias runtime endpoint names to physical limits.
    // Evaluate the policy and write both command bits together.
}
```

Complete the policy yourself. A dry input must suppress the command without resetting stored direction. A physical endpoint should change direction before the command is returned for that iteration.

3. Host-test sequences

Test complete sequences, not only isolated truth-table rows. Include reset, right travel, right-endpoint reversal, a dry pause, resume toward the left, left-endpoint reversal, dual-endpoint fault, recovery, and M/P non-interference.

4. Compile and program

1. Select `NUCLEO_WB55RG`, the Mbed environment, and Wiper 2-Bit simulation.
2. Compile the application and run the host policy tests.
3. Program the board with both outputs initially low.
4. Compare `(move1, move2)` with physical blade direction throughout the live test.

Required test sequence

Test	Action	Expected command and behavior
Reset, dry	Initialize with rain off.	`00`; default direction is right.
Rain start	Enable rain between endpoints.	`01`; move physically right.
Right endpoint	Assert runtime `leftSensor`.	`10`; reverse physically left.
Dry pause	Disable rain while travelling left.	`00`; left direction is retained.
Dry resume	Re-enable rain away from endpoints.	`10`; left travel resumes.
Left endpoint	Assert runtime `rightSensor`.	`01`; reverse physically right.
Button isolation	Toggle M and P while dry and while raining.	Required commands remain unchanged.
Sensor fault	Assert both endpoint inputs.	`11`.
Fault recovery	Clear the contradiction.	Coherent endpoint or stored direction determines travel.

Deliverables

Submit the state diagram, policy and application sources, host and live test evidence, and a short explanation of the aliases and dry direction retention.

Optional extensions

- Give M or P a documented manual behavior without changing the rain core.
- Add transition logging outside the policy and a fault indicator with explicit acknowledgement.
- Schedule the polling step periodically while retaining atomic snapshots.

Completion checklist

- GPIO reads are separated from the host-testable policy.
- Runtime endpoint names are translated to physical aliases once per snapshot.
- Direction reverses at each physical endpoint and survives dry pauses.