

TI C2000 Toolbox C-HIL Simulation

Start Synchronization Problem and Proposed Solution

1. Introduction

In a Controller Hardware-in-the-Loop (C-HIL) setup, a real controller executes control firmware while being connected to the HIL device that emulates the physical plant (e.g., an electric motor, inverter, grid, or power electronics system).

A common but often underestimated issue in such setups is **synchronization between the controller execution and the HIL simulation runtime**. This document describes the problem, its consequences, and a robust solution based on an explicit simulation-start synchronization signal.

2. Problem description

2.1. Controller perspective

Once flashed and powered, the controller:

- Starts executing its control code immediately
- Runs periodic control loops (ISR-based loops)
- Maintains internal states such as:
 - Integrators (PI / PID controllers)
 - Observers or estimators
 - Filters
 - State machines

Importantly, **the controller has no inherent knowledge of whether the HIL simulation is currently running or not**.

2.2. HIL perspective

The HIL simulation:

- Can be started, stopped or reloaded independently of the controller

- Initializes plant states only at simulation start
 - Resets all the HIL device outputs to zeros
-

3. Root cause of the issue

Because the controller begins execution before (or independently of) the HIL simulation:

- Control loops execute while:
 - Feedback signals are invalid or constant
 - References are zero or uninitialized
- Error signals become non-zero
- Integrators and other stateful elements start accumulating values

By the time the HIL simulation starts:

- The controller is already in a **non-deterministic and unwanted internal state**
 - Integrators may be saturated or biased
 - Observers may have drifted
 - State machines may be in an incorrect phase
-

4. Observed effects

Typical symptoms include:

- Large current, torque, or voltage spikes at simulation start – often causing an [arithmetic overflow](#) flag to raise
- Long transient recovery times
- Immediate controller saturation or fault triggering
- Non-reproducible test results between simulation runs

These effects reduce:

- Test reliability
- Repeatability

- Confidence in controller validation results
-

5. Why are software-only workarounds insufficient

Some common attempts to mitigate the problem include:

- Initializing controller states only once at boot
- Adding delays before enabling control
- Checking for signal plausibility

However, these approaches fail because:

- The controller cannot reliably infer simulation state from analog values
- Simulation start timing is tool- and user-dependent
- Delays are fragile and non-deterministic

A **clear, explicit synchronization mechanism** is required.

6. Proposed solution: simulation start synchronization signal

6.1. Concept

Introduce a **dedicated synchronization signal** that explicitly informs the controller when the HIL simulation starts.

This signal is:

- Generated by the HIL device
- Aligned exactly with simulation start
- Routed to the controller as a digital signal (GPIO/serial communication)

The controller detects a **rising edge** of this signal and uses it as a trigger to:

- Reset integrators
- Reset observers and filters
- Reinitialize internal controller states
- Enable control execution

6.2. Signal characteristics

Recommended properties:

- Type: Digital (0/1)
- Direction: HIL → Controller
- Timing: Asserted synchronously with simulation start
- Detection: Rising edge or level-based latch

Example:

- SIM_RUNNING = 0 → simulation not running
 - SIM_RUNNING = 1 → simulation started
-

7. Controller-side implementation

On the controller side, following options are available:

7.1. Component-level reset

State reset mechanism can be implemented for a single component. Typically, this can be performed with following sequence:

1. **Enable reset input in component** with state (such as [PID controller](#)) by setting property *External reset* (tab *Reset*) to *rising* or *either*, or any value other than *none* (keep in mind that other options might require different GPIO polarity and/or control settings). Additional input will be created.
2. **Configure a GPIO pin** as digital input with [GPIO DI \(Generic\)](#) or [GPIO DI](#) component to sample HIL digital output (Fig. 1).

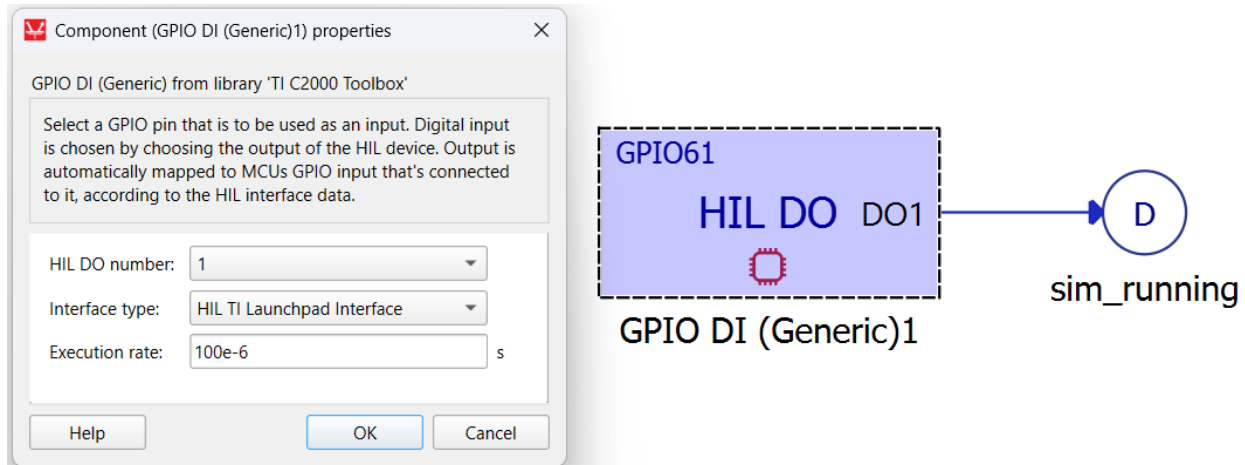


Figure 1. Configuring GPIO pin.

3. **Connect reset signal** (in this case signal *sim_running*) **to the reset input of the component** (for example, a [PID controller](#), reset input enabled in tab *Reset*) to reset its state (Fig 2.)

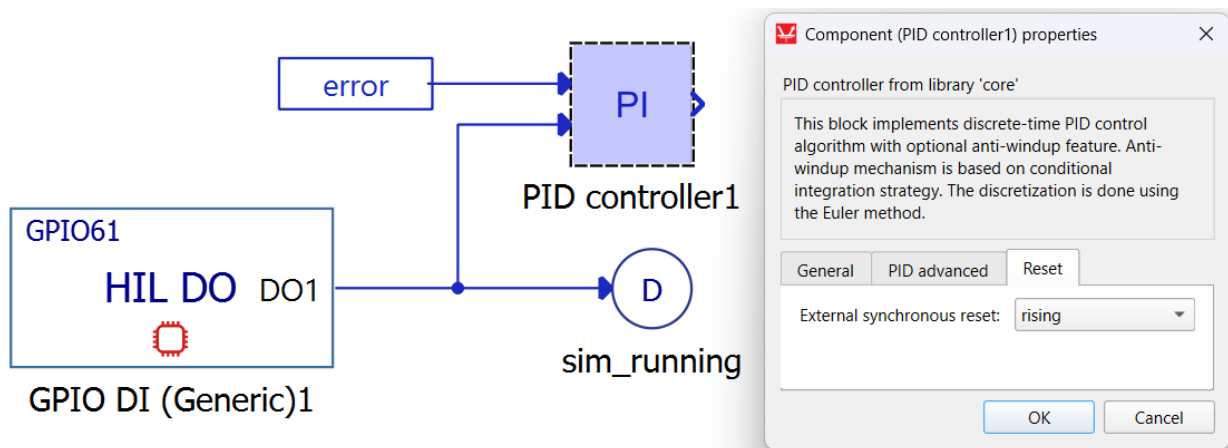


Figure 2. PID Controller - State reset input.

Controller will detect a rising edge on HIL DO on simulation start and will use that event to reset PID controller state.

7.2. State reset component

State reset component (Fig. 3) is used to reset *EVERY* state in exported code. Basically, the initialization function(s) of exported code will be executed – the same ones that are executed once the code is flashed. Most importantly, it will execute following actions:

- Clear PI/PID integrators

- Reset observers and filters
- Reset state machines
- *init_fnc* of every *C* function, etc.

Connect a signal that can be used for state reset to the component input. The principle described in section 7.1. is basically the same as this. In this case it is not required to enable *external reset* input in other components.

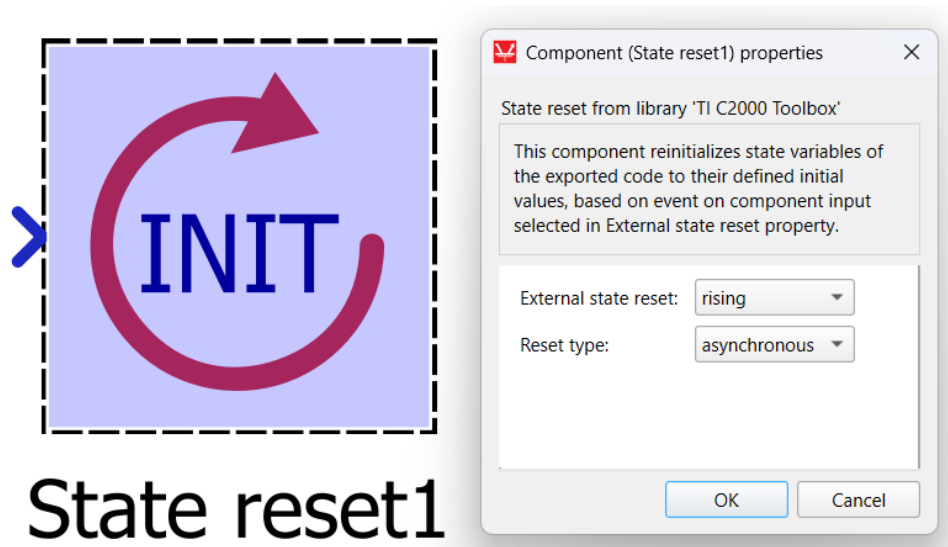


Figure 3. State reset component and its dialog.

7.2.1. Component properties

- **External state reset:** Select how the states are reset. The states can be reset on the rising edge, the falling edge, both the rising and the falling edges of the reset signal or it can stay in the initial state while the value of reset signal is not 0.
- **Reset type** - select the reset type between:
 - *asynchronous* – state is reset in the same cycle when the reset becomes active
 - *synchronous* – state is reset in the next cycle after the reset becomes active

7.2.2. Component inputs

- In1
 - Supported types: uint
 - Vector support: no

8. HIL-side implementation

On the HIL side:

- Drive the same digital output (as configured in previous steps) only when the simulation enters the "running" state (this can be done by enabling *SW Control* in [digital output settings](#) and setting value to 1, or by applying a constant to digital output in [Initial settings](#) component (Fig. 4)).

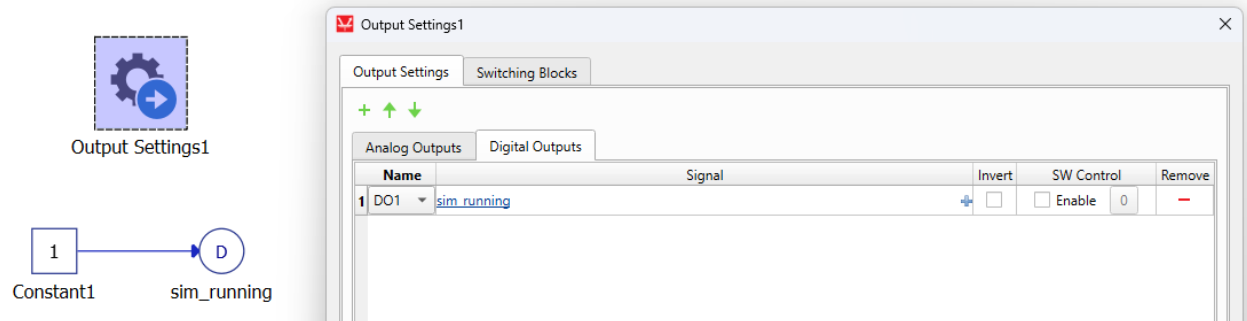


Figure 4. Reset signal - Initial settings

- Reset the signal when the simulation is stopped (this will be done automatically regardless of the signal applied to HIL outputs).

This makes the simulation start event explicit and reproducible.

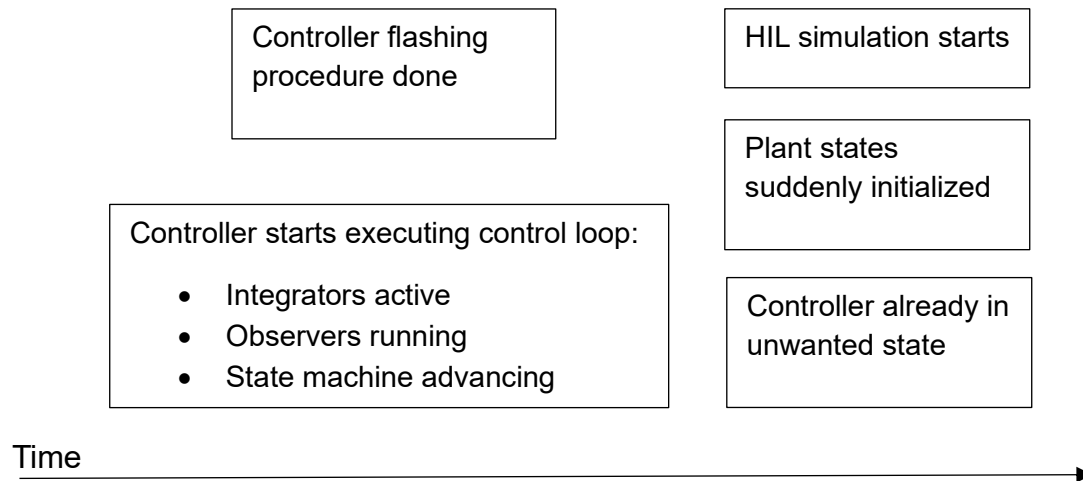
9. Benefits of the proposed approach

- ✓ Deterministic controller initialization
 - ✓ Repeatable simulation runs
 - ✓ Elimination of unwanted integrator wind-up
 - ✓ Clean and predictable startup transients
 - ✓ Clear separation between controller boot and simulation start
 - ✓ Scalable to multiple controllers and test cases
-

10. Timeline illustration (conceptual)

To better illustrate the problem and the proposed solution, the following conceptual timelines show the controller and HIL behavior over time.

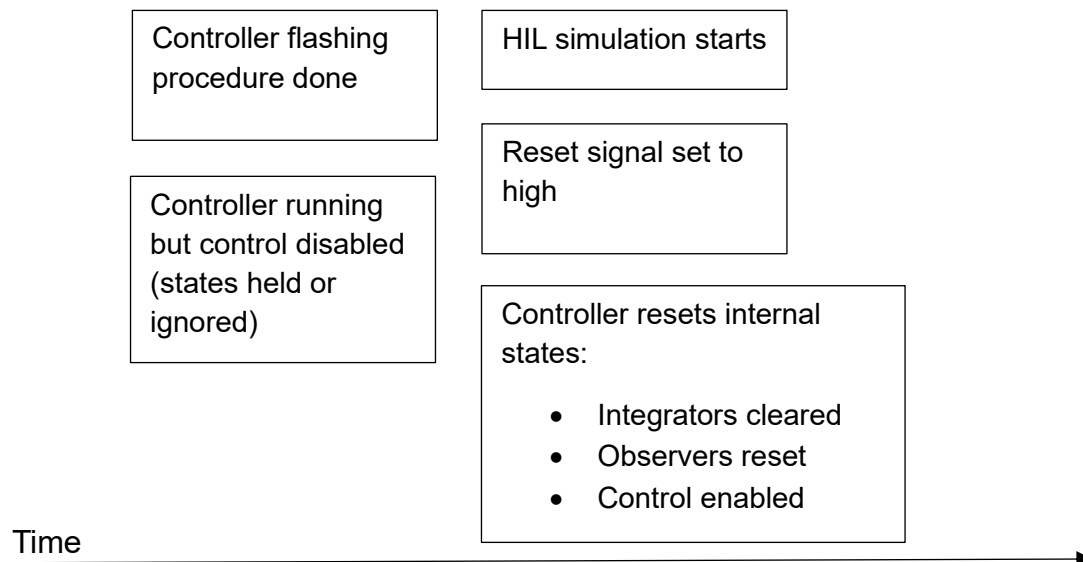
10.1. Without simulation start synchronization



Result:

- Integrator wind-up before simulation start
 - Large transient at first simulation step
 - Non-repeatable startup behavior
-

10.2. With simulation start synchronization signal



Result:

- Controller and plant start aligned
 - Clean and deterministic startup
 - Fully repeatable tests
-

11. Expected usage pattern

1. User flashes control code generated by the toolbox
2. Controller boots and waits for simulation start signal
3. User starts HIL simulation from the HIL environment
4. Simulation-aligned GPIO signal is asserted
5. Controller resets states and enables control

This workflow ensures that **every simulation run starts from a known and controlled initial condition**, independent of controller power cycling.