

TI C2000 Toolbox Task Scheduler Architecture

1. Purpose and scope

This document describes the task scheduling concept used in the toolbox-generated C control code. The scheduler provides a deterministic and efficient mechanism for executing multiple control tasks at different execution rates while maintaining strict timing relationships between them. The design is tailored for real-time control applications and controller hardware-in-the-loop (C-HIL) setups.

The scheduler is responsible **only for task timing and execution order**. All control algorithms are implemented in user-generated C code and are encapsulated within individual tasks.

2. Scheduler concept overview

The scheduler operates on a **task-based execution model** with the following core principles:

- Each task contains user-defined control logic written in C
- A separate *init* function is created for **each execution rate**
- A separate task (*step* function) is created for **each execution rate**
- All task execution rates must be **integer multiples of the fastest execution rate**
- A **single interrupt source** triggers the scheduler. This is specified in *Execution rate based on* property of [TI C2000 Setup](#) component (tab *Scheduling*). Scheduling interrupt must be enabled in corresponding component (ADC or ePWM) as well, unless the scheduler is CPU Timer based. Every other interrupt enabled in the code will not propagate to the scheduler (it will be ignored).
- Task execution is fully deterministic and repeatable

This structure ensures predictable timing behavior and avoids scheduling jitter or rate drift.

3. Initialization phase

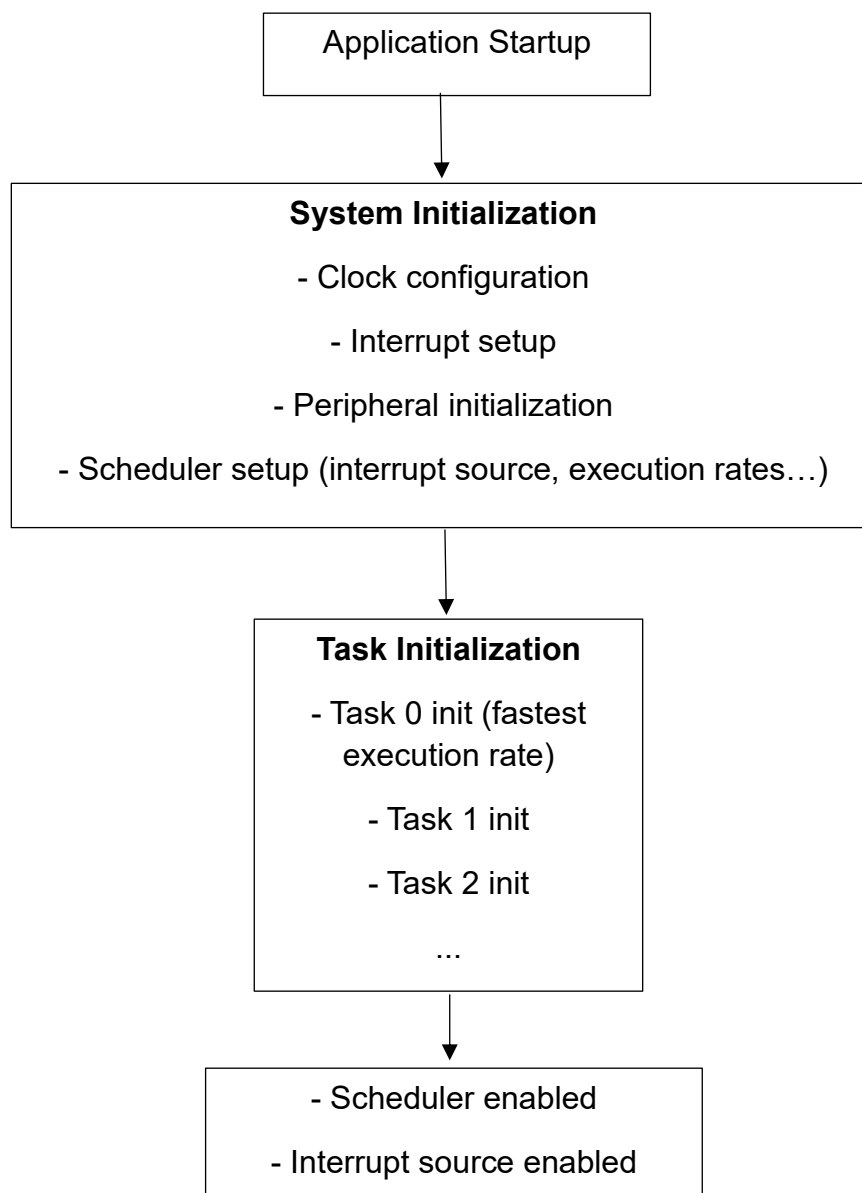
Before periodic task execution begins, the system performs a one-time initialization phase. This phase is executed at startup and is completed before the scheduler is enabled.

Initialization responsibilities

- Configuration of controller peripherals (ADC, PWM...)
- Initialization of internal controller states (integrators, filters, observers)
- Initialization of all tasks defined in the system

Once initialization is complete, the scheduler is activated and periodic task execution begins. Initialization code is **never re-executed** during runtime.

Initialization flow



4. Runtime scheduling mechanism

Interrupt-driven execution

The scheduler is driven by a **single periodic interrupt source**. Currently available interrupt sources are CPU timer, ePWM and ADC. This interrupt defines the **fastest execution rate** in the system and serves as the base time reference for all tasks.

At every interrupt occurrence:

- The scheduler is invoked
 - The fastest task is executed
 - A global execution counter is updated
-

5. Counter-based task activation

Task activation is controlled using an internal counter that increments on each scheduler interrupt. Slower tasks are executed when the counter reaches predefined thresholds corresponding to their execution rates.

Execution logic concept

- Fastest task: executed at every interrupt
- Slower tasks: executed when their integer multiple condition is met
- The medium-rate task executes every N fast-rate cycles
- The slow-rate task executes every M fast-rate cycles, and so on...
- All tasks remain phase-aligned to the fastest task

This approach guarantees that slower tasks execute at precise, deterministic intervals relative to the fastest task.

6. Summary

The task scheduler provides a robust and deterministic execution framework where:

- Tasks encapsulate user-defined control logic.

- Generated code is divided into separate *init* and *step* function for every *execution rate*.
- Execution rates are strictly defined and synchronized. **Slower execution rates must be an integer multiple of the fastest one.**
- A single interrupt source governs all timing. This is specified in *Execution rate based on* property of [TI C2000 Setup](#) component (tab *Scheduling*).
- Initialization and runtime behavior are clearly separated