# **EnduroSat Project - Technical Documentation**

## **device1.c**

* device1\_init()

Initializes and starts the master task using FreeRTOS's xTaskCreate().

* internal\_master\_task()

Implements the master task's logic: monitors slave state, maintains a fault counter, and triggers a reset if necessary.

## **device2.c**

* device2\_init()

Initializes and starts the slave task.

* internal\_slave\_task()

Implements the slave's logic, randomly simulating state changes and handling reset signals from the master.

## **main.c**

* main()

Initial entry point. Initializes semaphores, slave state, and starts FreeRTOS scheduler.

* slaveStateToString()

Converts SlaveState enum values to human-readable strings.

* masterStateToString()

Converts MasterState enum values to human-readable strings.

* vAssertCalled()

Custom assertion failure handler that disables interrupts and halts the system.

## **device1.h**

* MasterState enum

Defines MASTER\_IDLE, MASTER\_PROCESSING, MASTER\_ERROR — states for the master automaton.

* device1\_init()

Function prototype for initializing the master task.

## **device2.h**

* SlaveState enum

Defines SLAVE\_SLEEP, SLAVE\_ACTIVE, SLAVE\_FAULT — states for the slave automaton.

* device2\_init()

Function prototype for initializing the slave task.

## **main.h**

* device struct

Structure representing a slave device with ID and state.

* FAULT\_RESET\_THRESHOLD

Defines how many consecutive FAULTs trigger a reset request.

* Global Variables

Declares global mutexes, task handles, and flags for cross-module access.

* Helper Functions

Function prototypes for state-to-string conversions.

## **fake\_assert.h**

* vFakeAssert()

Custom assert handler that accepts a condition, filename, and line number and prints on failure.

## **hooks.c**

* vFakeAssert()

Prints failed assertion and terminates the program.

* vApplicationStackOverflowHook()

Called on stack overflow in any task, terminates with error message.

* vApplicationMallocFailedHook()

Called on memory allocation failure, logs and halts system.

* vApplicationGetIdleTaskMemory()

Provides static memory for the Idle Task.

* vApplicationGetTimerTaskMemory()

Provides static memory for the Timer Task.

* vConfigureTimerForRunTimeStats()

Stub for enabling run-time statistics tracking.

* portRECORD\_READY\_PRIORITY(), portRESET\_READY\_PRIORITY(), portGET\_HIGHEST\_PRIORITY()

Stubs used for managing and querying task priorities.

* vApplicationIdleHook()

Hook called when CPU is idle (sleeps for 1ms).

* vApplicationTickHook(), vApplicationDaemonTaskStartupHook()

Optional hooks defined but empty.

## **FreeRTOSConfig.h**

* configUSE\_16\_BIT\_TICKS

Determines the size of the tick count variable. Set to 0 for 32-bit (recommended).

* configUSE\_PREEMPTION

Enables preemptive multitasking (1) or cooperative (0).

* configUSE\_TIME\_SLICING

Enables round-robin scheduling for same-priority tasks.

* configTICK\_RATE\_HZ

System tick rate in Hz (e.g. 1000 = 1ms tick).

* configMAX\_PRIORITIES

Max task priority levels.

* configTOTAL\_HEAP\_SIZE

Total heap space (in bytes) used for dynamic allocation.

* configMAX\_TASK\_NAME\_LEN

Maximum number of characters allowed in task names.

* configMINIMAL\_STACK\_SIZE

Minimum stack size for Idle task.

* configTIMER\_TASK\_STACK\_DEPTH

Stack size for timer task.

* configCPU\_CLOCK\_HZ

Clock frequency of the CPU.

* configBYTE\_ALIGNMENT

Alignment used for memory allocations (usually 8).

* configSUPPORT\_DYNAMIC\_ALLOCATION

Allows kernel objects to be dynamically created at runtime.

* configSUPPORT\_STATIC\_ALLOCATION

Allows statically allocating memory for tasks and queues.

* INCLUDE\_vTaskDelay

Enable vTaskDelay API.

* INCLUDE\_vTaskDelete

Enable vTaskDelete API.

* INCLUDE\_vTaskSuspend

Enable suspend/resume APIs.

* INCLUDE\_xTaskGetSchedulerState

Enable API to get current scheduler state.

* INCLUDE\_vTaskDelayUntil

Enable periodic task delay API.

* configUSE\_STDIO

Redirects output to standard I/O.

* configPRINTF( x )

Wrapper for printf to allow RTOS-safe output.

* configASSERT( x )

Evaluates expressions and calls vAssertCalled() on failure.

* configCHECK\_FOR\_STACK\_OVERFLOW

Enable stack overflow detection.

* configUSE\_MALLOC\_FAILED\_HOOK

Enable malloc failure hook.

* configUSE\_IDLE\_HOOK

Allows a user-defined idle hook function.

* configUSE\_TICK\_HOOK

Allows a tick hook to be defined.

* configUSE\_DAEMON\_TASK\_STARTUP\_HOOK

Defines a startup hook for the daemon task.

* configUSE\_TASK\_NOTIFICATIONS

Enable lightweight task notifications.

* configUSE\_TIMERS

Enables timer service task.

* configUSE\_MUTEXES

Enable use of mutexes.

* configUSE\_COUNTING\_SEMAPHORES

Enable counting semaphores.

* configUSE\_QUEUE\_SETS

Enable queue sets functionality.

## **port.c**

* xPortStartScheduler()

Initializes critical section and creates fake timer thread to simulate RTOS tick.

* prvTimerThread()

Simulates periodic tick interrupt calling task switching.

* vPortStartFirstTask()

Launches task threads manually (simulated).

* portTaskWrapper()

Task entry wrapper to convert function pointer.

* pxPortInitialiseStack()

Initializes and starts a new thread to simulate a task.

* vPortYield()

Triggers Windows thread yield.

* vPortInitializeCriticalSection()

Initializes critical section structure for mutual exclusion.

* vPortEnterCritical(), vPortExitCritical()

Enter and exit critical sections.

* vPortEndScheduler()

Stops the simulated tick thread.

## **portmacro.h**

* portSTACK\_TYPE, portBASE\_TYPE, etc.

Defines type mappings used across the port (e.g., StackType\_t, BaseType\_t).

* portTICK\_PERIOD\_MS

Defines tick period in milliseconds.

* portYIELD()

Calls vPortYield().

* Critical section macros

ENTER/EXIT/ENABLE/DISABLE interrupts via critical section emulation.

* portTASK\_FUNCTION\_PROTO / portTASK\_FUNCTION

Defines task prototypes and functions for consistency.

* portNOP()  
  No-operation macro.