

# EMBEDDED SYSTEMS

BASED ON CORTEX-M4 AND THE RENESAS  
SYNERGY PLATFORM

2020

PROF. DOUGLAS RENAUX, PHD  
PROF. ROBSON LINHARES, DR.  
UTFPR / ESYSTECH

RENESAS ELECTRONICS CORPORATION

# 14 – RTOS – REAL-TIME OPERATING SYSTEM

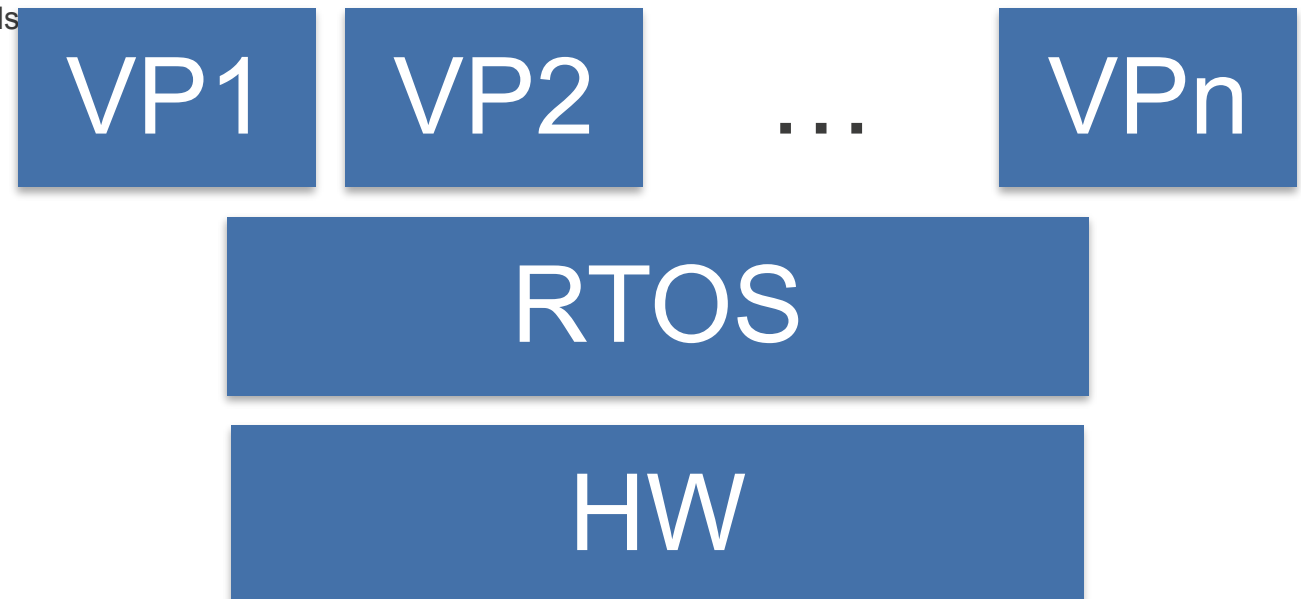
---

- Introduction
- Thread Management
- Inter-thread communication and synchronization
- Timing Services
- Memory Management

# RTOS – INTRODUCTION

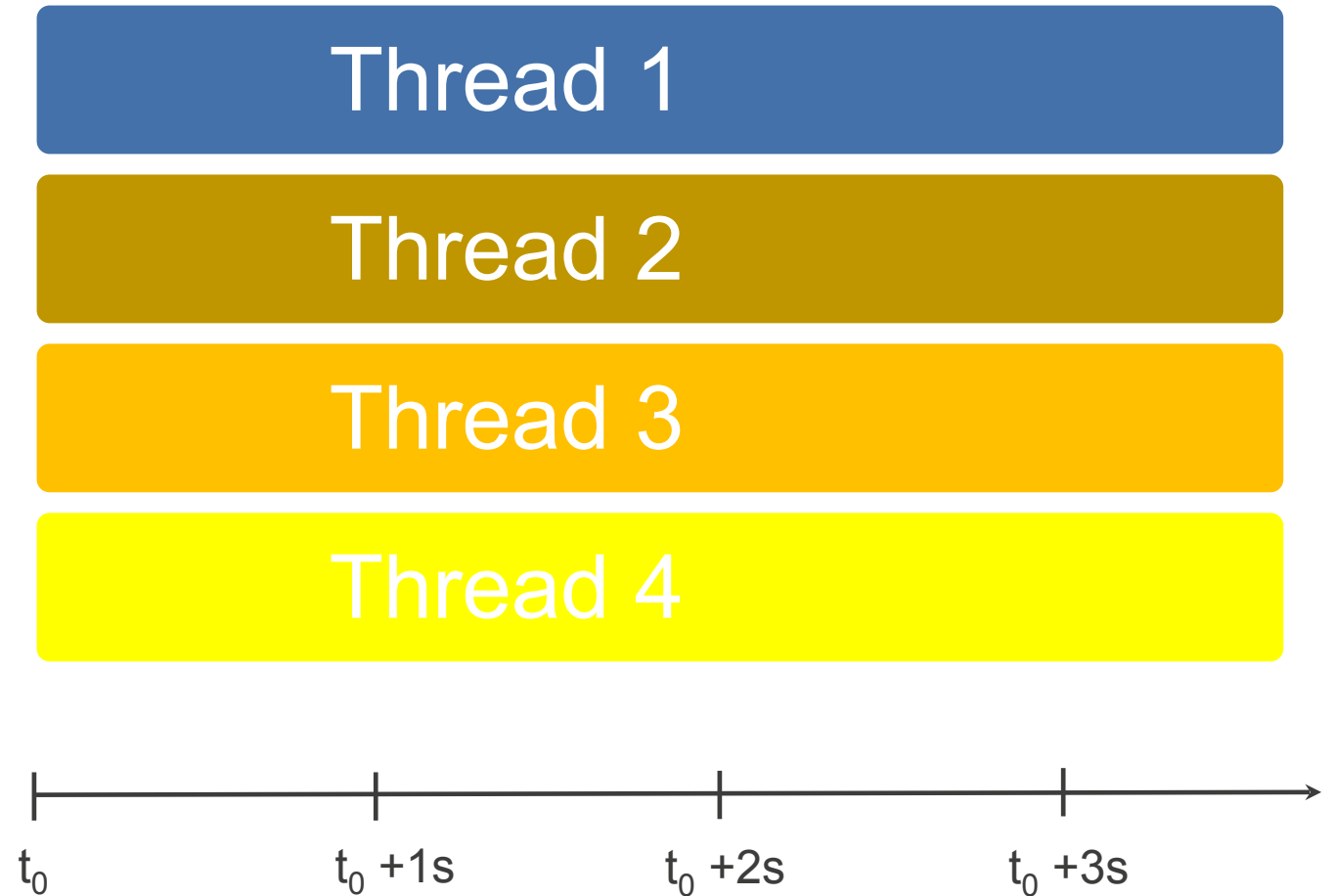
---

- An RTOS is a layer of Software between the hardware and the application software that allows the implementation of concurrent tasks implemented by multithreading.
- The RTOS manages the hardware, sharing resources among multiple “virtual processors”: VP1 to VPn so that each thread has the illusion of owning a processor and stack while sharing global data, heap, code area and peripherals
- The RTOS implements sharing of the processor via context switching and provides several important functionalities for the threads: timing, inter-thread communication and synchronization, and thread management.



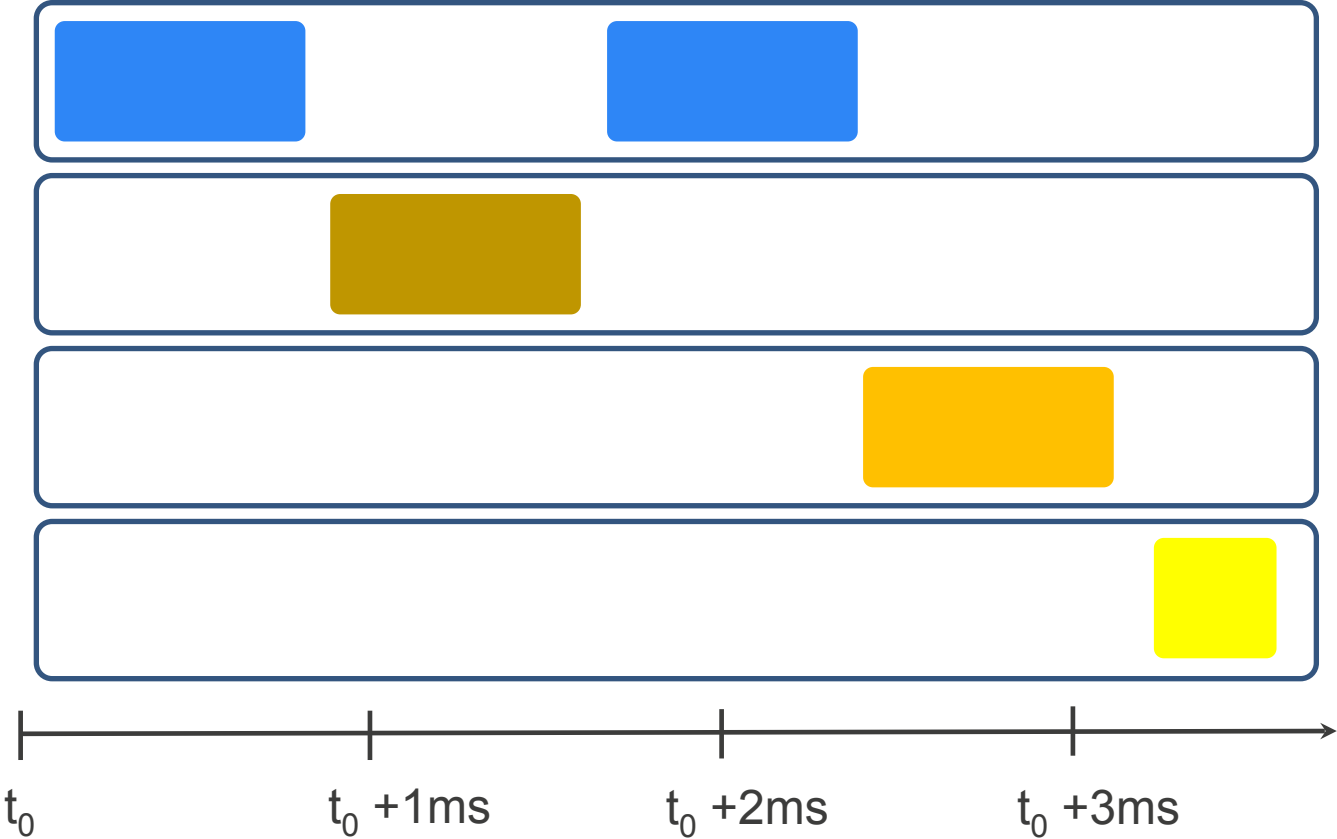
# THREAD EXECUTION – LARGE GRAIN VIEW

Observing at a time scale of seconds, one has the impression that all threads execute in parallel.



# THREAD EXECUTION – FINE GRAIN VIEW

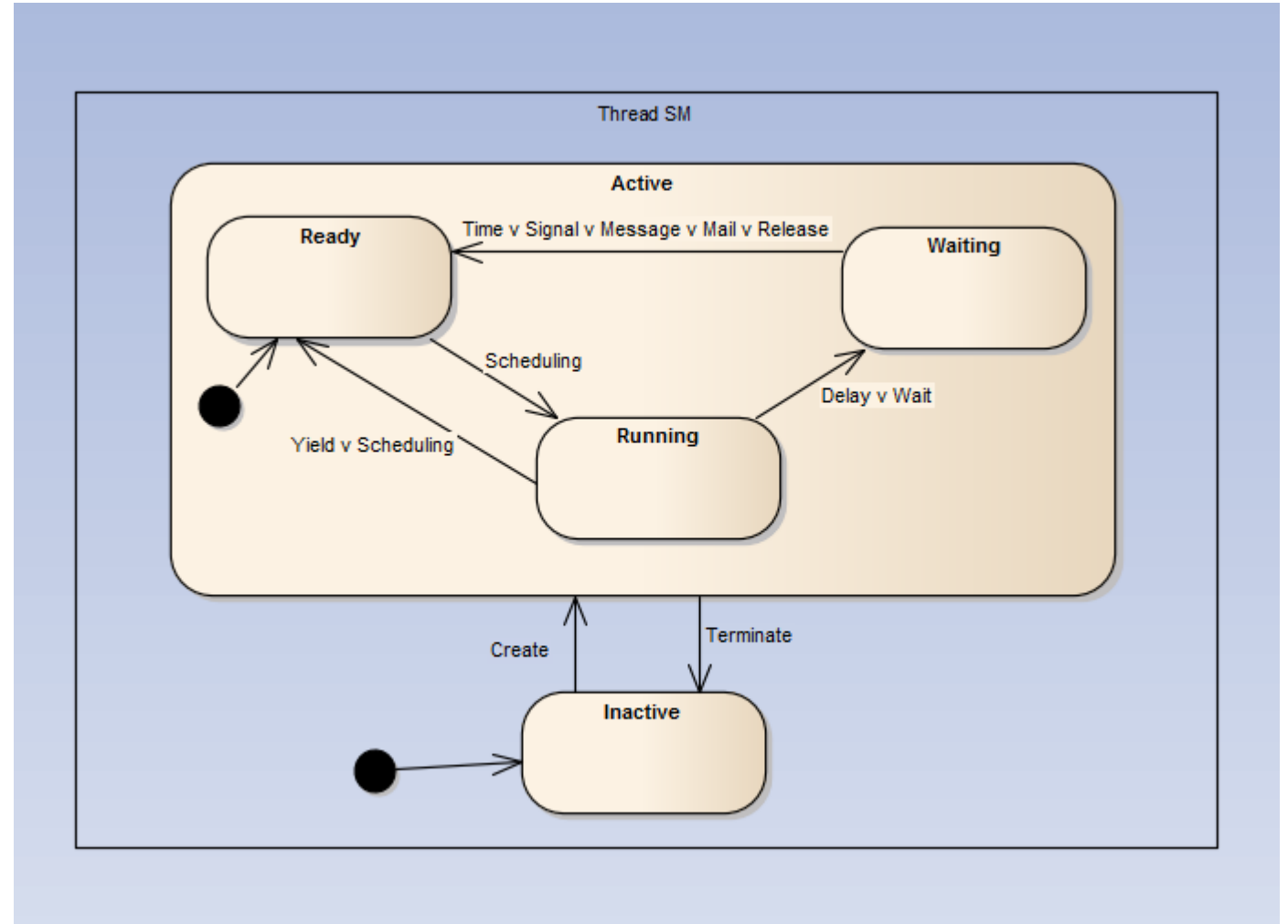
Observing on a fine grain scale (milliseconds) it is possible to realize that the tasks share the processor as the RTOS switches among them the utilization of the CPU.



# TASK STATES

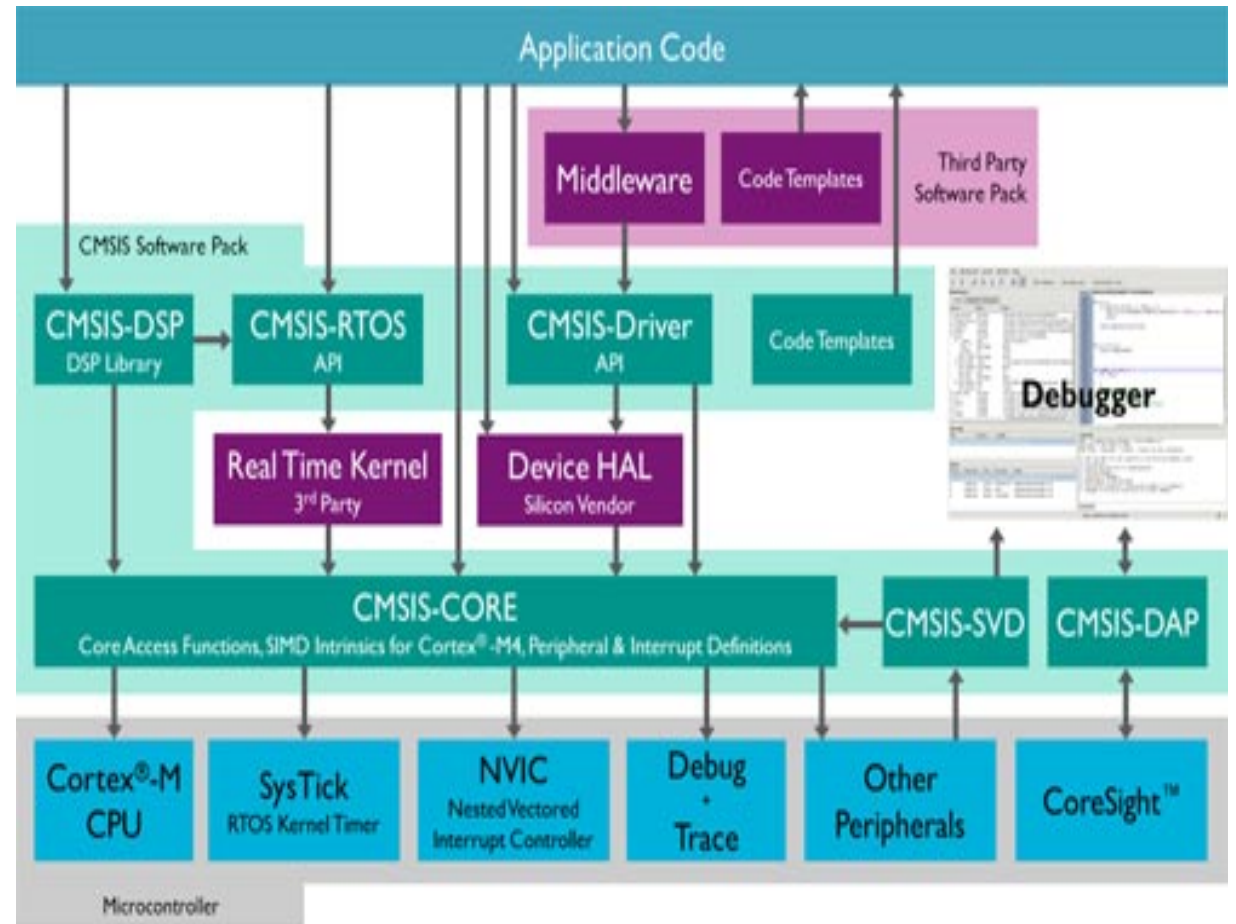
A typical state diagram of a task in a multithreaded environment:

- **Ready:** tasks that is willing to use the processor and is expected to be scheduled.
- **Running:** the state of the task that is using the processor.
- **Waiting:** tasks whose execution is blocked due due to time, synchronization or communication.



# CMSIS (CORTEX MICROCONTROLLER SOFTWARE INTERFACE STANDARD)

- v 1.0 (2008)
  - Drivers API
- v 2.0
  - DSP Lib
- v 3.0 (2012)
  - API RTOS
  - DAP (Debug)
- V 4.0 (2014)
  - CMSIS-Driver
  - CMSIS-Pack
- V 5.0 (2016)
  - CMSIS-RTOS v2



As of March, 2020, CMSIS is at version 5.6

# CMSIS-RTOS DOCUMENTATION

Available [online](#)

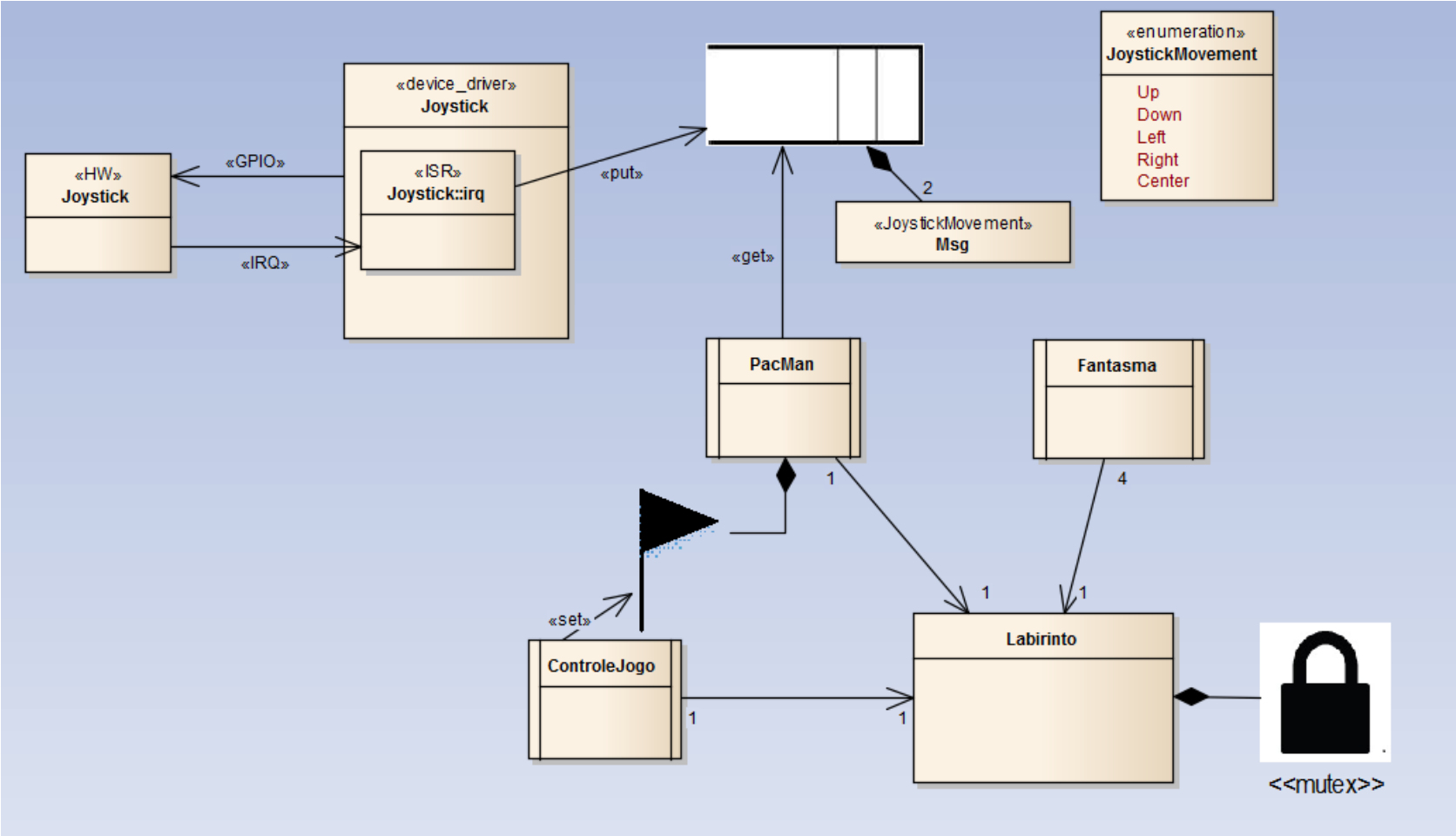
The screenshot displays the CMSIS-RTOS2 documentation website. At the top, the CMSIS logo is on the left, followed by the title 'CMSIS-RTOS2 Version 2.1.2' and the subtitle 'Real-Time Operating System: API and RTX Reference Implementation'. Below this is a navigation bar with tabs for 'General', 'CMSIS-Core(A)', 'CMSIS-Core(M)', 'Driver', 'DSP', 'RTOS v1', 'RTOS v2' (which is active), 'Pack', 'SVD', 'DAP', and 'Zone'. Underneath, there are sub-tabs for 'Main Page', 'Usage and Description', and 'Reference' (which is active). A left sidebar contains a tree view of the documentation structure, with 'Reference' selected. The main content area is titled 'Reference' and contains the text 'Here is a list of all modules:'. Below this is a table listing various modules and their descriptions.

Module	Description
<b>CMSIS-RTOS2 API</b>	Describes the C function interface of CMSIS-RTOS API v2
<b>Kernel Information and Control</b>	Provides version/system information and starts/controls the RTOS Kernel
<b>Thread Management</b>	Define, create, and control thread functions
<b>Thread Flags</b>	Synchronize threads using flags
<b>Event Flags</b>	Synchronize threads using event flags
<b>Generic Wait Functions</b>	Wait for a certain period of time
<b>Timer Management</b>	Create and control timer and timer callback functions
<b>Mutex Management</b>	Synchronize resource access using Mutual Exclusion (Mutex)
<b>Semaphores</b>	Access shared resources simultaneously from different threads
<b>Memory Pool</b>	Manage thread-safe fixed-size blocks of dynamic memory
<b>Message Queue</b>	Exchange messages between threads in a FIFO-like operation
<b>Definitions</b>	Constants and enumerations used by many CMSIS-RTOS functions
<b>Flags Functions Error Codes</b>	Constants used by <b>Thread Flags</b> and <b>Event Flags</b> to return error codes
<b>OS Tick API</b>	Provides a low level API between an device agnostic RTOS implementation and specific p
<b>RTX5 Specific API</b>	This section describes CMSIS-RTOS RTX5 specifics
<b>Macros</b>	RTOS macros
<b>Structs</b>	RTOS structs
<b>Functions</b>	RTOS functions
<b>Event functions</b>	RTOS Event Recorder functions
<b>Memory Functions</b>	Events generated memory functions
<b>Kernel Functions</b>	Events generated by kernel functions
<b>Thread Functions</b>	Events generated by thread functions

CMSIS code is available in GITHUB



# PLANNING A MULTITHREADED APPLICATION



# CMSIS-RTOS: OVERVIEW 1/4

---

## Kernel Information and Control

- [osKernelInitialize](#) : Initialize the RTOS kernel.
- [osKernelStart](#) : Start the RTOS kernel.
- [osKernelRunning](#) : Query if the RTOS kernel is running.
- [osKernelSysTick](#) \$ : Get RTOS kernel system timer counter.
- [osKernelSysTickFrequency](#) \$ : RTOS kernel system timer frequency in Hz.
- [osKernelSysTickMicroSec](#) \$ : Convert microseconds value to RTOS kernel system timer value.

## Thread Management

- [osThreadCreate](#) : Start execution of a thread function.
- [osThreadTerminate](#) : Stop execution of a thread function.
- [osThreadYield](#) : Pass execution to next ready thread function.
- [osThreadGetId](#) : Get the thread identifier to reference this thread.
- [osThreadSetPriority](#) : Change the execution priority of a thread function.
- [osThreadGetPriority](#) : Obtain the current execution priority of a thread function.

# CMSIS-RTOS: OVERVIEW 2/4

---

## Generic Wait Functions

- osDelay : Wait for a specified time.
- osWait \$ : Wait for any event of the type Signal, Message, or Mail.

## Timer Management \$

- osTimerCreate : Define attributes of the timer callback function.
- osTimerStart : Start or restart the timer with a time value.
- osTimerStop : Stop the timer.
- osTimerDelete : Delete a timer.

# CMSIS-RTOS: OVERVIEW 3/4

---

## Signal Management

- [osSignalSet](#) : Set signal flags of a thread.
- [osSignalClear](#) : Reset signal flags of a thread.
- [osSignalWait](#) : Suspend execution until specific signal flags are set.

## Mutex Management \$

- [osMutexCreate](#) : Define and initialize a mutex.
- [osMutexWait](#) : Obtain a mutex or Wait until it becomes available.
- [osMutexRelease](#) : Release a mutex.
- [osMutexDelete](#) : Delete a mutex.

## Semaphore Management \$

- [osSemaphoreCreate](#) : Define and initialize a semaphore.
- [osSemaphoreWait](#) : Obtain a semaphore token or Wait until it becomes available.
- [osSemaphoreRelease](#) : Release a semaphore token.
- [osSemaphoreDelete](#) : Delete a semaphore.

# CMSIS-RTOS: OVERVIEW 4/4

---

## Memory Pool Management \$

osPoolCreate : Define and initialize a fix-size memory pool.

osPoolAlloc : Allocate a memory block.

osPoolCAlloc : Allocate a memory block and zero-set this block.

osPoolFree : Return a memory block to the memory pool.

## Message Queue Management \$

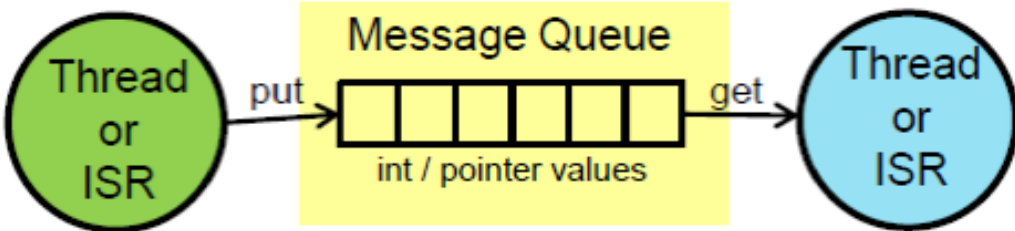
- osMessageCreate : Define and initialize a message queue.
- osMessagePut : Put a message into a message queue.
- osMessageGet : Get a message or suspend thread execution until message arrives.

## Mail Queue Management \$

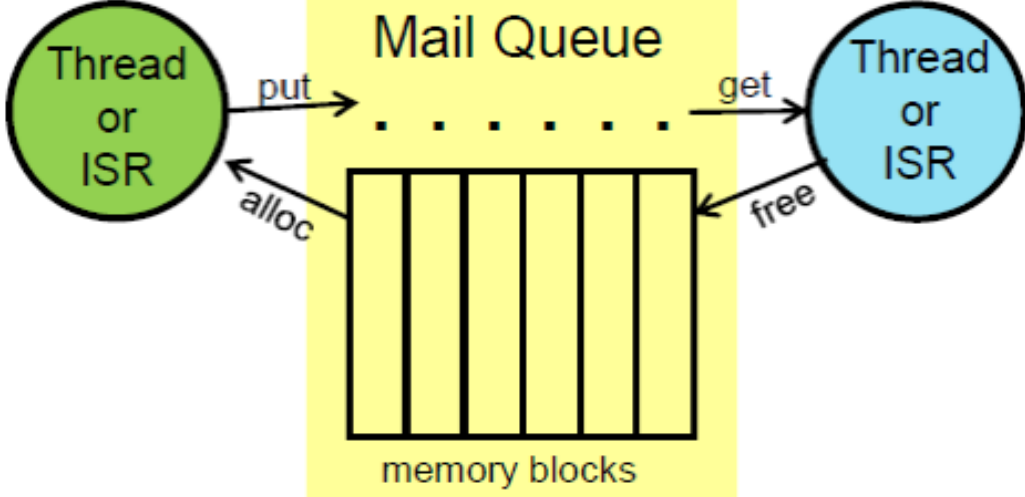
- osMailCreate : Define and initialize a mail queue with fix-size memory blocks.
- osMailAlloc : Allocate a memory block.
- osMailCAlloc : Allocate a memory block and zero-set this block.
- osMailPut : Put a memory block into a mail queue.
- osMailGet : Get a mail or suspend thread execution until mail arrives.
- osMailFree : Return a memory block to the mail queue.

# CMSIS-RTOS – INTER-TASK COMMUNICATIONS

## Message: Integer or Pointer



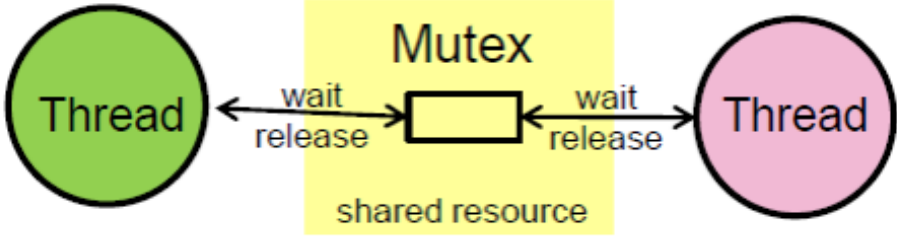
## Mail: Memory Blocks



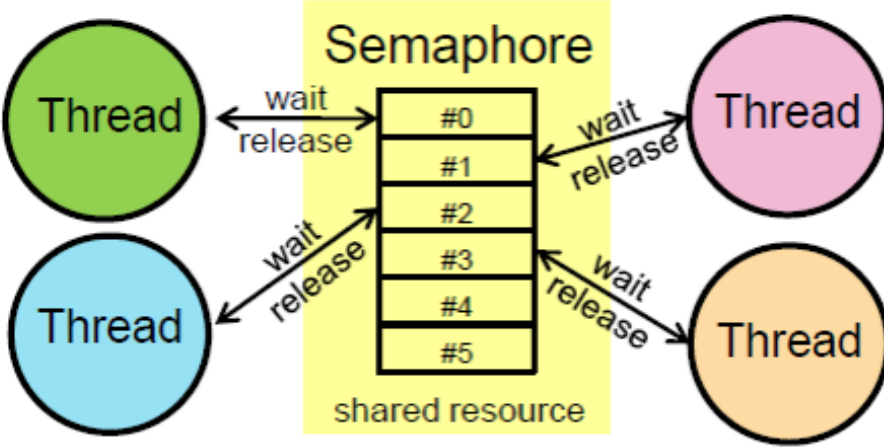
Source: ARM, CMSIS documentation

# CMSIS-RTOS – INTER-TASK COMMUNICATIONS

## Mutex: Synchronization



## Semaphore: Shared Access



Source: ARM, CMSIS documentation

---

[Renesas.com](https://www.renesas.com)