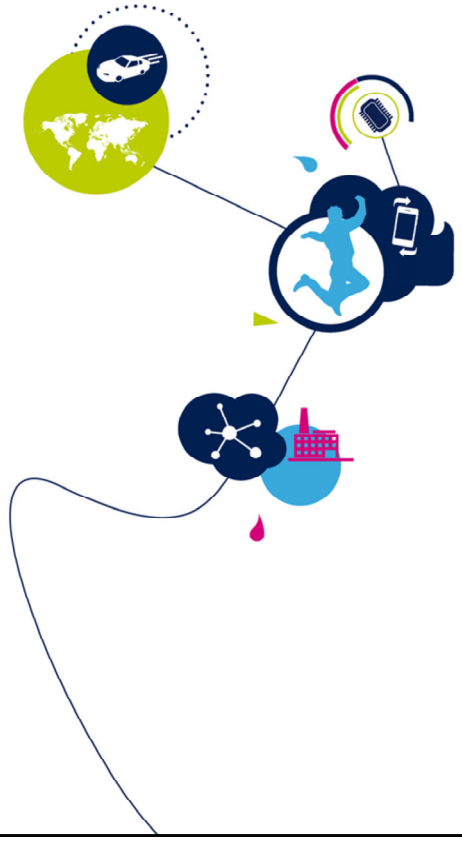
STM32WB - HSEM

Hardware Semaphore

Revision 1.0



life.augmented

Hello, and welcome to this presentation of the STM32 hardware semaphore (HSEM) module. It covers the main features of the module which is used to manage the access permissions and synchronization of resources shared between multiple processes.

Overview

- Manages access permissions and synchronization between
  - Different processes running on a same CPU
  - Different CPUs
- 32 Semaphores
- Two locking mechanisms
  - 2-step write, read back lock
  - 1-step read lock
- Semaphore-free interrupt generation

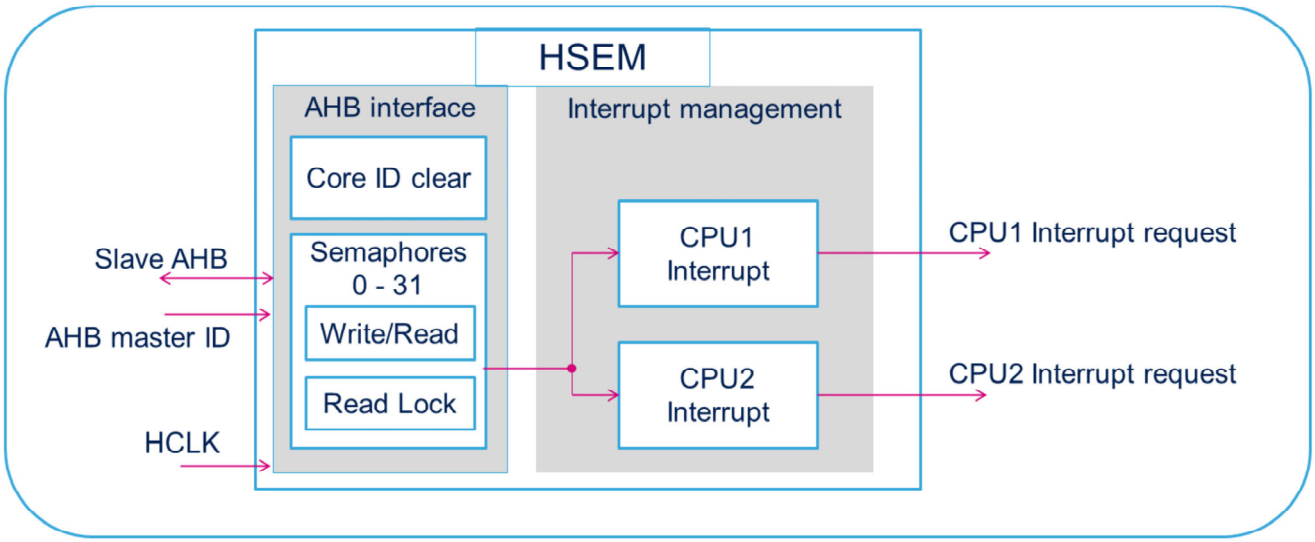
Application benefits

- Prevent access conflicts between shared resources
- Ensure synchronization between processes
- Non-blocking semaphore handling

life.augmented

The HSEM module integrated inside STM32 microcontrollers provides semaphores used to synchronize processes and manage access permissions for shared resources. This module provides a 2-step lock mechanism and a fast 1-step lock mechanism. Applications benefit from process synchronization and sharable resources via non-blocking interrupt-based semaphores.

Block diagram

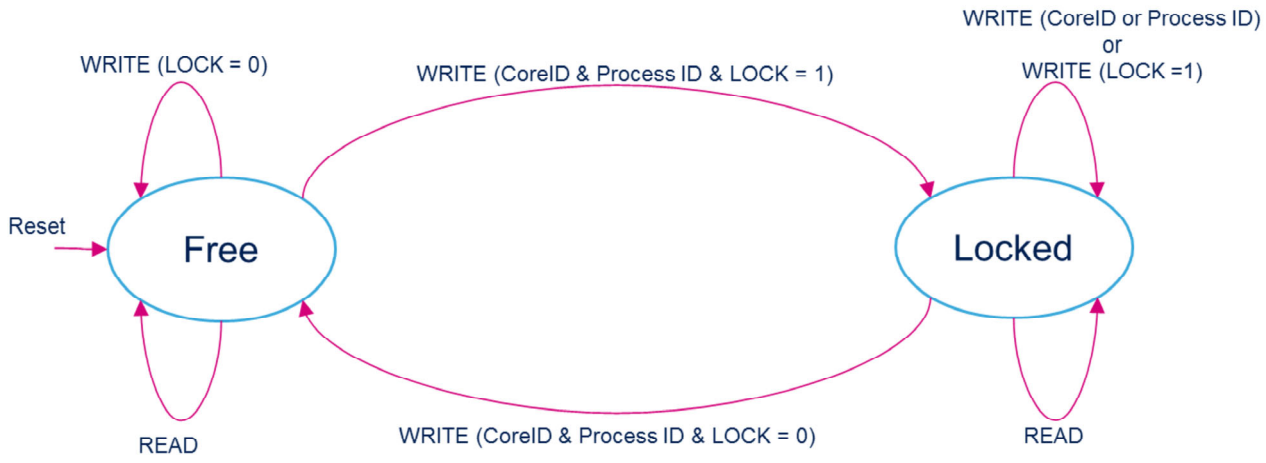


The HSEM module is located on the AHB bus.
It consists of an “AHB interface” containing the semaphores and an “Interrupt management” block handling the interrupt distribution.
A dedicated interrupt is provided for each CPU. Each CPU has its own enable, status, mask and clear register for the semaphores.
Each semaphore consists of 2 registers, a Write/Read register, used to write lock the semaphore in the 2-step procedure and read back the semaphore status. The Write/Read register is also used to free a semaphore.
The Read Lock register is used to read lock the semaphore in the 1-step procedure.
The AHB bus master ID is used to identify which CPU is accessing the semaphore. This ID is stored in the semaphore when locking, and can be read back from the semaphore status as Core ID.

In STM32WB microcontrollers, CPU1 uses Core ID 0x03, and CPU2 has Core ID 0x01.

Semaphore procedure

2-step write lock mechanism

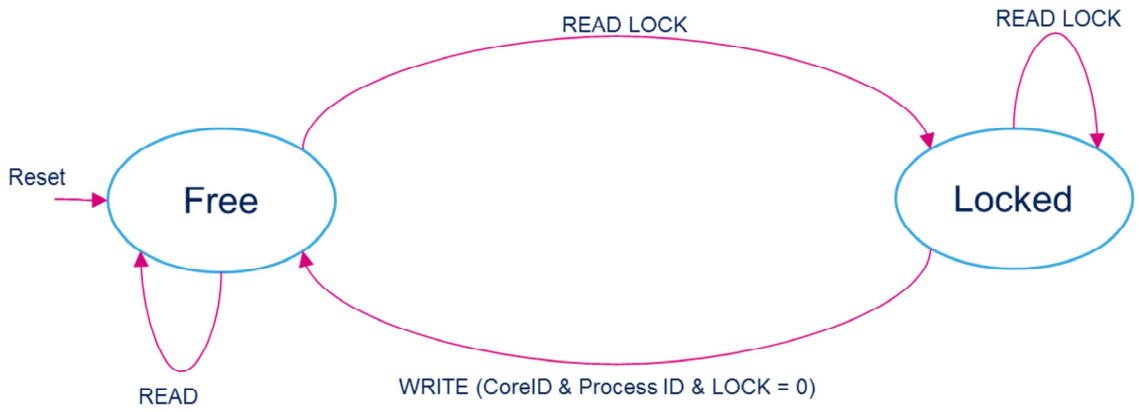


In the 2-step write lock procedure a “free” semaphore will be locked by writing ‘1’ to the LOCK bit in the semaphore Write/Read register. The Core ID and Process ID used during the write will be stored in the semaphore. A process has to check that the semaphore is locked by it, by reading back the Write/Read register. If the read back semaphore Core ID and Process ID matches the one written by the process locking the semaphore it is locked by it. If the Core ID or Process ID doesn’t match, the semaphore has been locked by another process.
A locked semaphore can only be unlocked by writing the LOCK bit to ‘0’ with the corresponding Core ID and Process ID.
Writing the Core ID or the Process ID or writing the LOCK bit with ‘1’ will keep the semaphore locked.

Semaphore procedure

5

1-step read lock mechanism



In the 1-step read lock procedure, a “free” semaphore will be locked by reading the semaphore’s Read lock register. The Core ID used during the read will be stored in the semaphore during the read cycle. When the semaphore Core ID value read by the CPU matches the one from the CPU and the Process ID = 0x0000, the semaphore is locked by the CPU. In the 1-step read lock procedure there is NO Process ID. The Process ID when locked by a 1-step read lock procedure will read 0x0000. If the Core ID doesn’t match or the Process ID is different from 0x0000, the semaphore has been locked by another CPU or process.

A locked semaphore can only be unlocked by writing the LOCK bit to ‘0’ with the corresponding Core ID and Process ID.

The 2-step and 1-step lock procedures can be used concurrently. In this case, the 2-step lock procedure must

not use Process ID value 0x0000.

Semaphore procedure

General Core ID clear

- All semaphores locked by a CPU may be cleared via the semaphore clear register.
  - Allows to clear locked semaphores when a CPU is no longer functioning correctly.
- Write Core ID and Key value.

life.augmented

In the case where locked semaphores from a malfunctioning CPU are to be cleared, this can be done by writing a Key value in the HSEM_KEYR register and writing the Key value and the Core ID in the HSEM_CR register. It will clear all semaphores locked by the corresponding Core ID, a semaphore get free interrupt will be generated when enabled.

HSEM Interrupts

Interrupt event	Description
INT0_SEM	Semaphore getting free interrupt to CPU1
INT1_SEM	Semaphore getting free interrupt to CPU2

life.augmented

Here is an overview of HSEM interrupt events. When a semaphore is freed, an interrupt can be generated to a CPU. Each CPU has its own set of semaphore enable (IER) and status (ISR) states before the masking (MISR) and clear (ICR) registers.

Low-power modes

Mode	Description
Run	Active. Peripheral interrupts cause a CPU to exit CSleep and CStop modes.
Sleep	Frozen.
Stop	Frozen in STOP0, STOP1 and STOP2 modes
Standby	Powered-down. The peripheral must be reinitialized after exiting system Standby mode.

life.augmented

Here is an overview of the peripheral status at specific low-power configuration modes.
The HSEM is not able to change state in system Sleep and Stop modes.
In Standby mode, the content of the HSEM is lost.
The HSEM can only be changed when the system is in Run mode.
The HSEM is able to interrupt and wake up a CPU in CRun, CSleep, and CStop modes.

Application examples

- Manage shared resources
  - RCC, PWR, AES, RNG, ...
  - Lock semaphore before peripheral access.
  - Free semaphore once done.
- Synchronize processes
  - Process A locks semaphore when waiting for input, and got to sleep.
  - Process B, when input available, locks and frees an other semaphore, which will wake up Process A.
  - Processor A frees its semaphore.

life.augmented

Semaphores can be used to handle:
1: Access to shared resources, i.e. this allows to read, modify and write registers in a protected way or to share functions like the AES encryption engine, random number generator (RNG), etc...
2: Synchronization between processes, i.e. used by a process to wait for input from another process.

Related peripherals

- This is a list of peripherals related to the HSEM controller. Please refer to these trainings for more information if needed.
  - Reset and clock control (RCC)
  - Interrupts (NVIC)
  - Asynchronous Event and Interrupt Control (AIEC)

life.augmented

Here is a list of peripherals related to the HSEM module. Users should be familiar with all the relationships between these peripherals to correctly configure and use the HSEM module.