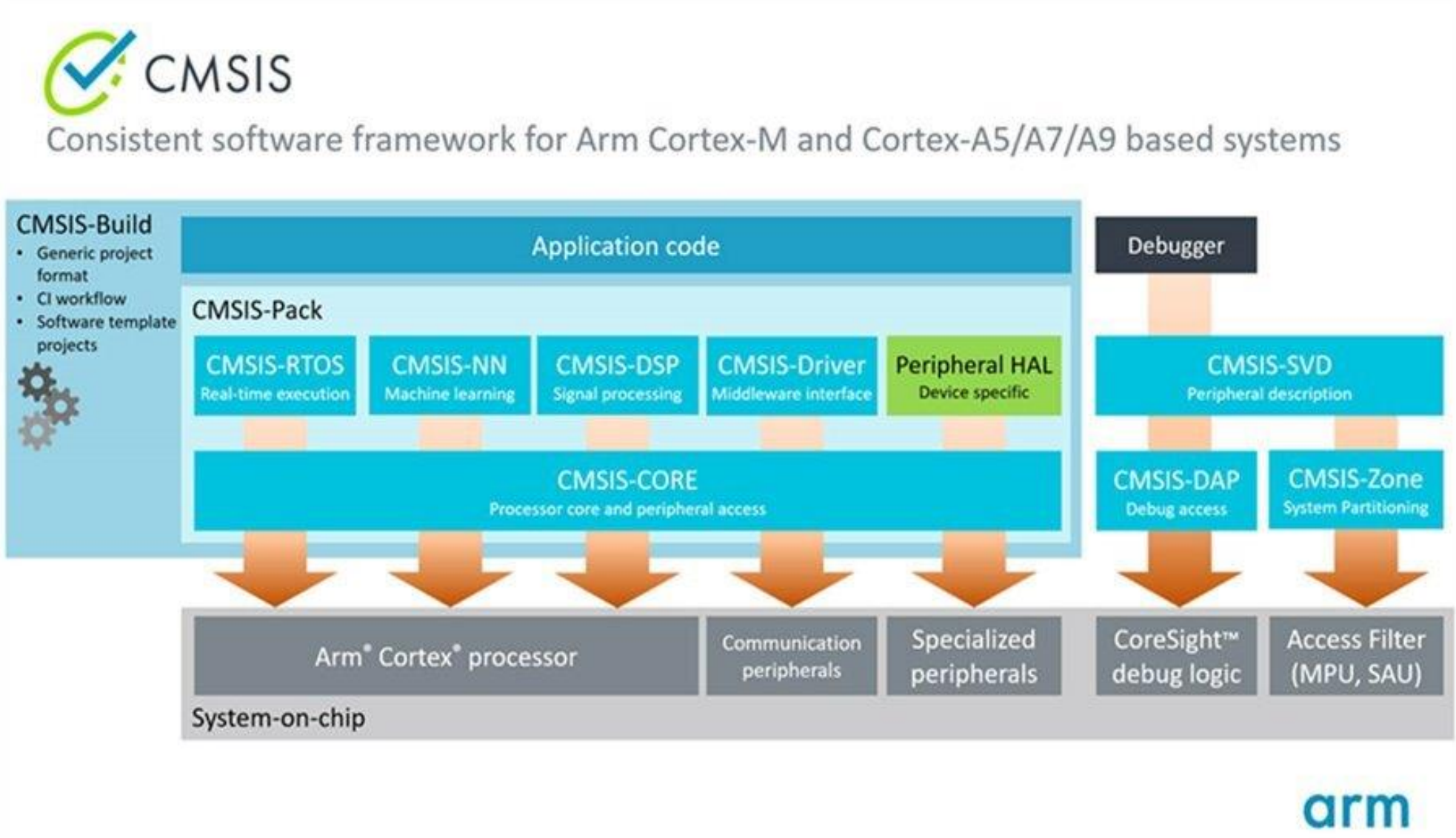
CMSIS Real Time Operating System V2

https://arm-software.github.io/CMSIS_5/develop/RTOS2/html/index.html

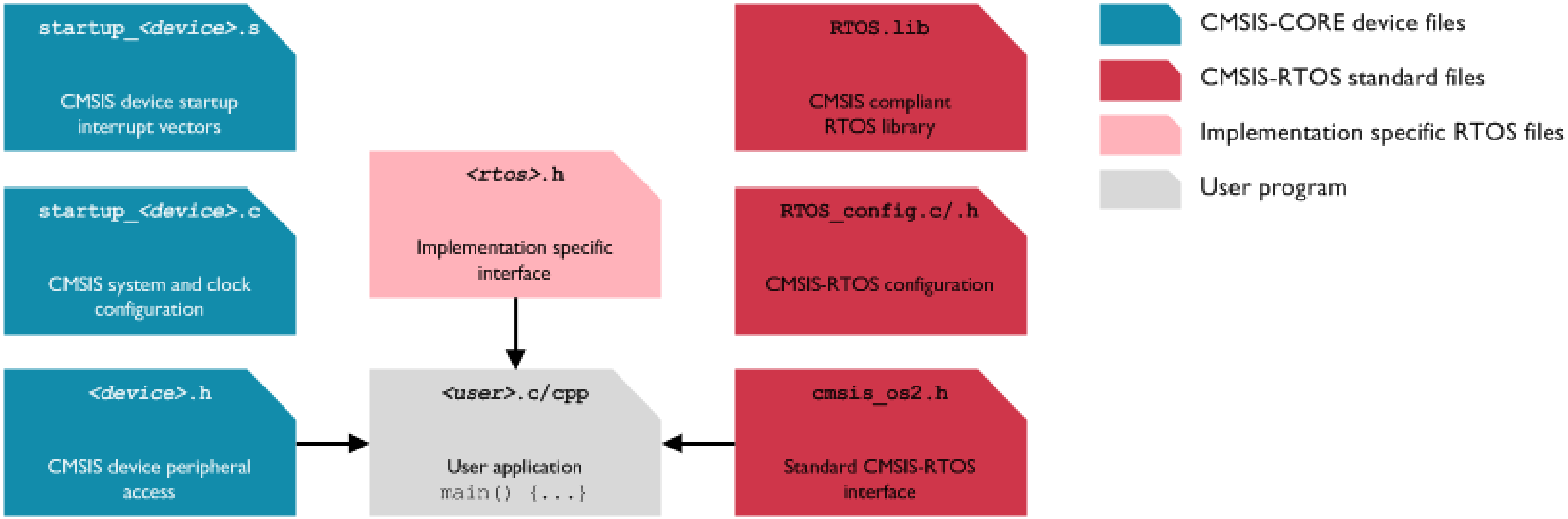
CMSIS

- The Cortex Microcontroller Software Interface Standard (CMSIS)
  - is a vendor-independent hardware abstraction layer for microcontrollers that are based on Arm Cortex processors.
  - CMSIS defines generic tool interfaces and enables consistent device support.
    - CMSIS provides interfaces to processor and peripherals, real-time operating systems, and middleware components.
  - The CMSIS software interfaces simplify software reuse, reduce the learning curve for microcontroller developers, and improve time to market for new devices.
- Open source – License Apache 2.0
  - https://arm-software.github.io/CMSIS_5/General/html/cm_revisionHistory.html
  - Current version 5.9 – also every module has a revision number
  - Legacy version 4

CMSIS Components



CMSIS-RTOS2 File Structure (v2)



Code Example

```
/*----------------------------------------------------------------------
* CMSIS-RTOS 'main' function template
*----------------------------------------------------------------------*/
#include "RTE_Components.h"
#include CMSIS_device_header
#include "cmsis_os2.h"
/*----------------------------------------------------------------------
* Application main thread
*----------------------------------------------------------------------*/
void app_main (void *argument) {
// ...
for (;;) {}
}
int main (void) {
// System Initialization
SystemCoreClockUpdate();
// ...
osKernelInitialize(); // Initialize CMSIS-RTOS
osThreadNew(app_main, NULL, NULL); // Create application main thread
osKernelStart(); // Start thread execution
for (;;) {}
}
```

CMSIS-RTOS2 (RTOS v2) API

- **Thread Management** allows you to define, create, and control threads.
- **Interrupt Service Routines (ISR)** can call some CMSIS-RTOS functions.
  - When a CMSIS-RTOS function cannot be called from an ISR context, it rejects the invocation and returns an error code.
- Thread Communication and Synchronization
  - Three event types support communication between multiple threads and/or ISR:
    - **Thread Flags**: may be used to indicate specific conditions to a thread.
    - **Event Flags**: may be used to indicate events to a thread or ISR.
    - **Messages**: can be sent to a thread or an ISR. Messages are buffered in a queue.
  - **Mutex** Management and **Semaphore**s are incorporated.
- CPU time can be scheduled with the following functionalities:
  - A timeout parameter is incorporated in many CMSIS-RTOS functions to avoid system lockup. When a timeout is specified, the system waits until a resource is available or an event occurs. While waiting, other threads are scheduled.
  - The **osDelay** and **osDelayUntil** functions put a thread into the WAITING state for a specified period of time.
  - The **osThreadYield** provides co-operative thread switching and passes execution to another thread of the same priority.
- **Timer Management** functions are used to trigger the execution of functions.

Inter-Thread Communication

- Thread flag – for thread synchronization
  - Each thread has a pre-allocated 32-bit thread flag object.
  - A thread can wait for its TFs to be set by threads/interrupts.
- Event flag – for thread synchronization
  - Similar to thread flags, except dynamically created
- Semaphore – control access to common resource
  - Semaphore object contains tokens (“counting” semaphore)
  - Thread can request a token (put to sleep if none available)
- Mutex – mutual exclusion locks
  - “lock” a resource to use it, and unlock it when done
  - Kernel suspends threads that need the resource until unlocked
- Message Queue (Mail Queue eliminated in RTOS2)
  - Queue is a first-in/first-out (FIFO) structure
  - “Message” is an integer or a pointer to a message frame
  - Suspend thread if “put” to full queue or “get” from empty queue

Thread Flags

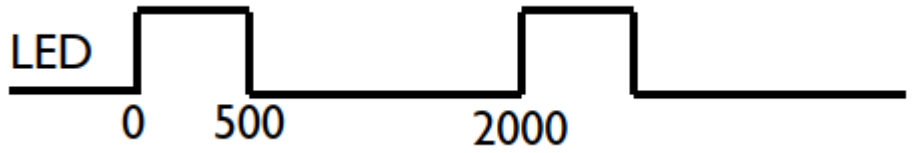
- Thread flags not “created” – a 32-bit word with 31 thread flags; exists automatically within each thread.

- One thread sets TFs in another thread (addressed by its thread ID)

Thread Flags API

- osThreadFlagsSet(tid, flags) – set TFs of thread tid
  - flags = int32_t; each “1” bit in “flags” sets the corresponding TF
  - Example: flags=0x8002 => set/clear TF #15 and TF #1
- osThreadFlagsWait(flags, option, timeout)
  - Wait for TFs corresponding to “1” bits in “flags” to be set
  - Option = osFlagsWaitAny or osFlagsWaitAll = wait for any or all of the flags
  - Timeout = 0 (check and return), osWaitForever, or time T
  - Return 32-bit value of flags (and then clear them)
    - osFlagsErrorTimeout if TFs are set before timeout T
    - osFlagsErrorResource if TFs not set and timeout = 0
    - osFlagsErrorUnknown unspecified error (not called from correct context)
- osThreadFlagsClear(tid, flags) – clear TFs of thread, return current flags set
- osThreadFlagsGet( ) – return flags currently set in this thread

CMSIS-RTOS thread flags example




```
//Thread 1
void ledOn (void constant *argument) {
        for (;;) {
            LED_On(0);
            osThreadFlagsSet(tid_ledOff, 0x0001); //signal ledOff thread
            osDelay(2000);
        }
}
// Thread 2
void ledOff (void constant *argument) {
        for (;;) {
            // wait for signal from ledOn thread
            osThreadFlagsWait(0x0001, osFlagsWaitAny, osWaitForever);
            osDelay(500);
            LED_Off(0);
        }
}
```

Thread Flag Example – Thread3 must wait for signals from both Thread1 and Thread2

```
// Thread Flag Example – Thread3 must wait for signals from both Thread1 and Thread2
#include "cmsis_os2.h"
osThreadId_t tid1; //three threads
osThreadId_t tid2;
osThreadId_t tid3;
void thread1 (void *argument) {
        while (1) {
                osThreadFlagsSet(tid3, 0x0001);   /* signal thread 3 */
                ….
void thread2 (void *argument) {
        while (1) {
                osThreadFlagsSet(tid3, 0x0002);   /* signal thread 3 */
                ….
void thread3 (void *argument) {
uint32_t flags;
        while (1) {
        //wait for signals from both thread1 and thread2
        flags = osThreadFlagsWait(0x0003, osFlagsWaitAll, osWaitForever);
        … //continue processing
```

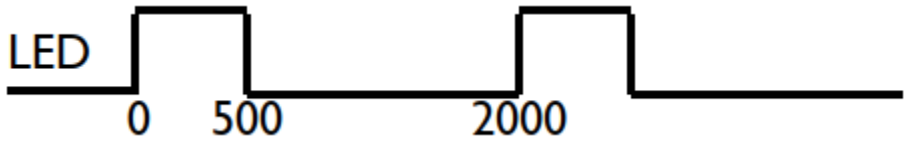
CMSIS-RTOS2 Event Flags

- Each “signal” has up to 31 “event flags” (bits 30-0 of the signal word)
- Event Flags should be created when needed
  - Similar to Thread Flags, but Event Flags do not “belong” to any thread
  - Wait (in BLOCKED state) for an event flag to be set
  - Set/Clear one or more event flags

Event Flags API

- osEventFlagsId_t evt_id;
  evt_id = osEventFlagsNew(*attr) – create & initialize event flags
  - NULL argument for default values (or pointer to osEventFlagsAttr_t structure)
  - Return event flags id (evt_id)
- osEventFlagsSet(evt_id, flags) – set EFs in evt_id
  osEventFlagsClear(evt_id, flags) – clear EFs of evt_id
  - flags = int32_t; each “1” bit in “flags” sets/clears the corresponding EF
  - Return int32_t = flags after executing the set/clear (or error code)
- osEventFlagsWait(evt_id, flags, options, timeout)
  - Wait for EFs corresponding to “1” bits in “flags” to be set, or until timeout
  - Options – osFlagsWaitAny or osFlagsWaitAll (any or all of the indicated flags)
  - Return current event flags or error code
    - osFlagsErrorTImeout if awaited flags not set before timeout
    - osFlagsErrorResouce awaited flags not set and timeout = 0

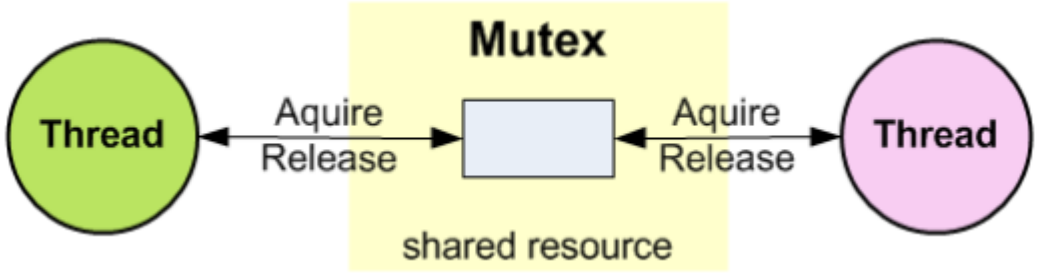
Event flags example




```
osEventFlagsId_t led_flag;
void main_app (void constant *argument) {
        led_flag = osEventFlagsNew(NULL); //create the event flag
}
void ledOn (void constant *argument) {
        for (;;) {
            LED_On(0);
            osEventFlagsSet(led_flag, 0x0001); //signal ledOff thread
            osDelay(2000);
        }
}
void ledOff (void constant *argument) {
    for (;;) { // wait for signal from ledOn thread
        osEventFlagsWait(led_flag, 0x0001, osFlagsWaitAny, osWaitForever);
        osDelay(500);
        LED_Off(0);
        }
}
```

Mutual Exclusion (MUTEX)

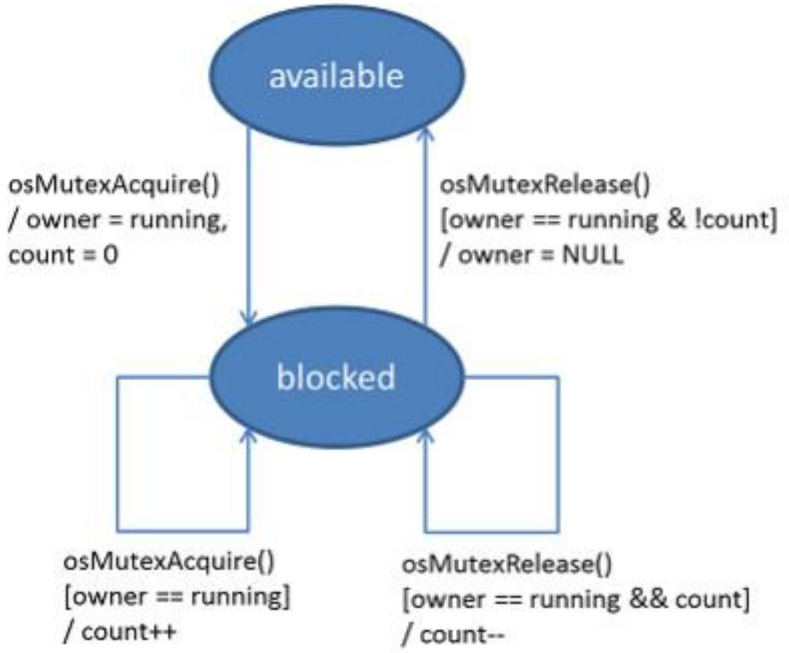
- Binary semaphore
  - Provide exclusive access to a resource



osMutexAcquire(mutex_id, timeout)
osMutexRelease(mutex_id)

- Limit access to shared resource to one thread at a time.

- Special version of a “semaphore”

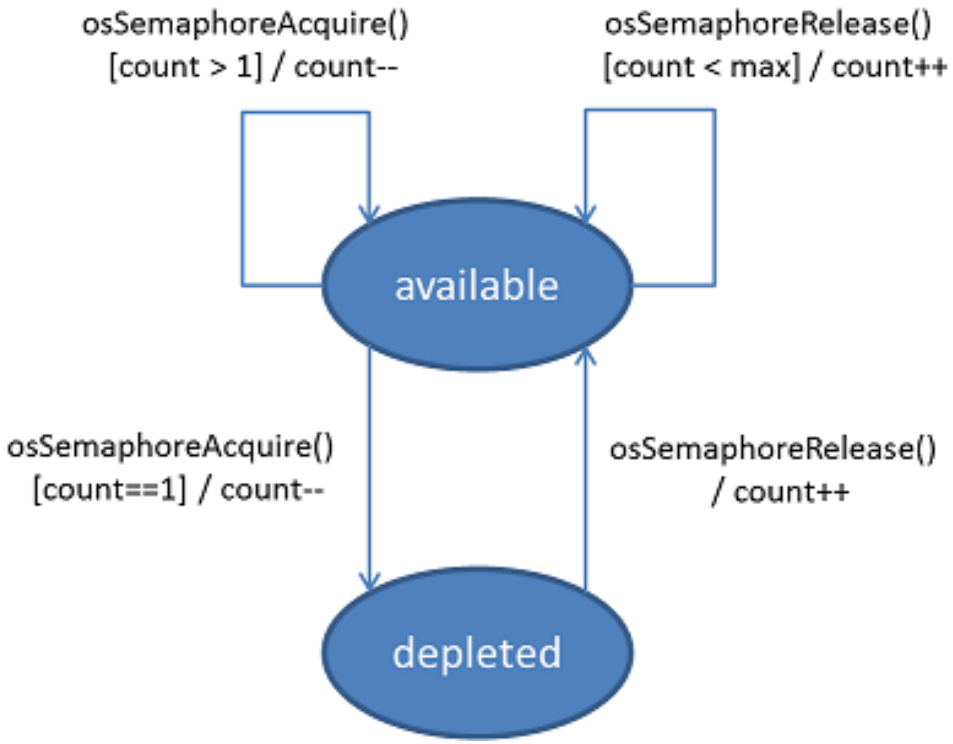


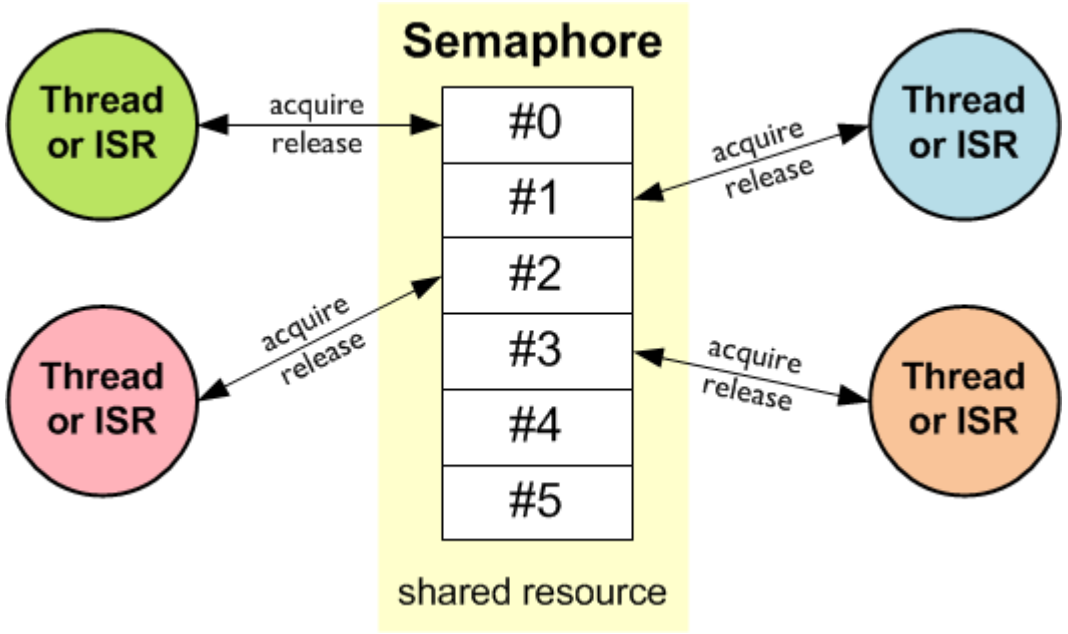
Mutual Exclusion (MUTEX) API

- osMutexId_t m_id; //MUTEX ID
  - m_id = osMutexNew(attr); //create MUTEX obj
  - attr = osMutexAttr_t structure or NULL for default
- status = osMutexAcquire(m_id, timeout);
  - Wait until MUTEX available or until time = “timeout”
  - timeout = 0 to return immediately
  - timeout = osWaitForever for infinite wait
  - “status” = osOK if MUTEX acquired
  - osErrorTimeout if not acquired within timeout
  - osErrorResource if not acquired when timeout=0 specified
- status = osMutexRelease(m_id); //release the MUTEX
  - status = osOK if released, osErrorResource if invalid operation (not owner)

Semaphores

- Counting Semaphore
- Allow up to t threads to access a resource
- Permit fixed number of threads/ISRs to access a pool of shared resources.
- Initialize with max # of “tokens”.





Semaphore API

- osSemaphoreId s_id; //semaphore ID
- s_id = osSemaphoreNew(max_tokens, init_tokens, attr);
  - Create s1; set max and initial #tokens
  - attr osSemaphoreAttr_t structure or NULL for defaults
- status = osSemaphoreAcquire(s_id, timeout);
  - Wait until token available or timeout
  - status = osOK if token obtained (#tokens decremented)
  - osErrorTimeout if token not obtained before timeout
  - osErrorResouce if token not obtained and timeout=0
- status = osSemaphoreRelease(s_id);
  - Release token
  - status = osOK if token released (#tokens incremented)
  - osErrorResouce if max token count reached
  - osErrorParameter if s_id invalid

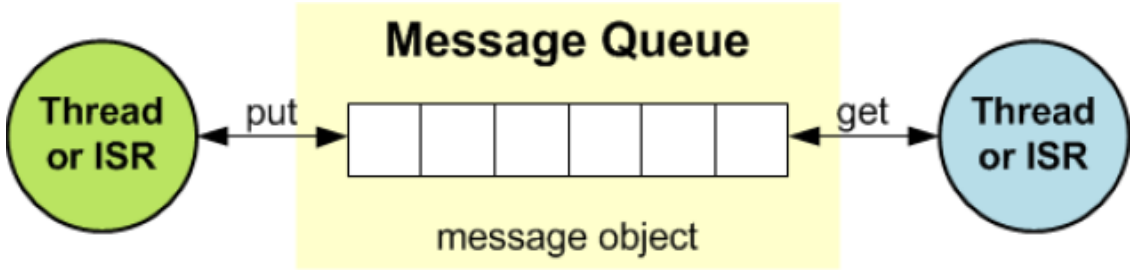
CMSIS-RTOS semaphore example

```
osSemaphoreId_t multiplex_id;

void thread_n (void) {
        multiplex_id =
        osSemaphoreNew(3U, 3U, NULL);
        while(1) {
                osSemaphoreAcquire(multipl
                ex_id, osWaitForever);
                // do something
                osSemaphoreRelease(multipl
                ex_id);
                }
}
```

```
#define BUFFER_SIZE 10U

osSemaphoreId_t empty_id = osSemaphoreNew(BUFFER_SIZE, BUFFER_SIZE,
NULL);

osSemaphoreId_t filled_id = osSemaphoreNew(BUFFER_SIZE, 0U, NULL);

void producer_thread (void) {

while(1) {
        osSemaphoreAcquire(empty_id, osWaitForever);
        // produce data
        osSemaphoreRelease(filled_id);
        }

}

void consumer_thread (void) {

while(1){
        osSemaphoreAcquire(filled_id, osWaitForever);
        // consume data
        osSemaphoreRelease(empty_id);
        }

}
```

Message queues

- “Message” = information to be sent



- osMessageQueuePut(mq_id, *msg_ptr, msg_prio, timeout)
- osMessageQueueGet(mq_id, *msg_ptr, msg_prio, timeout)
- osMessageQueueGetCapacity(mq_id) - max #msgs in the queue
- osMessageQueueGetMsgSize(mq_id) - max msg size in memory pool
- osMessageQueueGetCount(mq_id) - # queued msgs in the queue
- osMessageQueueGetSpace(mq_id) - # available slots in the queue
- osMessageQueueReset(mq_id) - reset to empty

Message queues API

- osMessageQueueId q_id; // ID of queue object
- q_id = osMessageQueueNew(msg-count, msg-size, attr);
  - Create and initialize a message queue, return queue ID
  - Specify: max #msgs, max msg size, attributes (or NULL for defaults)
- status = osMessageQueuePut(q_id, msg-ptr, msg-priority, timeout);
  - Add message to queue; wait for “timeout” if queue full
  - msg-ptr = pointer to message data structure
  - Status = osOK : msg was put into the queue
  - = osErrorResource : not enough space for msg
  - = osErrorTimeout : no memory available at timeout
- status = osMessageQueueGet(q_id, msg-ptr, msg-priority, timeout);
  - Get msg from queue and put in *msg-ptr; wait for “timeout” if no message
  - Status = osOK : no msg available and timeout=0
  - = osEventTimeout : no message available before timeout
  - = osEventMessage : msg received (“status” is a “union” structure)

```
/* Message Queue creation & usage example */
// message object data type
typedef struct {
      uint8_t Buf[32];
      uint8_t Idx;
} MSGQUEUE_OBJ_t;
// message queue id
osMessageQueueId_t mid_MsgQ;
// thread creates a message queue for 12 messages
int Init_MsgQueue (void) {
      mid_MsgQ = osMessageQueueNew(12, sizeof(MSGQUEUE_OBJ_t), NULL);
      ….

}
```

/* Message Queue Example Continued */

```
void Thread1 (void *argument) {    // this threads sends data to Thread2
MSGQUEUE_OBJ_t msg;
while (1) {
        ; // Insert thread code here...
        msg.Buf[0] = 0x55; // data to send
        msg.Idx = 0; // index of data in Buf[]
        osMessageQueuePut (mid_MsgQ, &msg, 0, NULL); // send the message
        osThreadYield (); // suspend thread
} }
void Thread2 (void *argument) {   //This thread receives data from Thread1
MSGQUEUE_OBJ_t msg;
osStatus_t status;
while (1) {
        ; // Insert thread code here...
        status = osMessageQueueGet (mid_MsgQ, &msg, NULL, NULL); // wait for message
        if (status == osOK) {
                ; // process data in msg.Buf[msg.Idx]
}   } }
```

References

- **CMSIS-RTOS2 Documentation**
  - https://arm-software.github.io/CMSIS_5/RTOS2/html/index.html
- **Generic RTOS Interface**
  - https://arm-software.github.io/CMSIS_5/RTOS2/html/genRTOS2IF.html
- **CMSIS-RTOS C API v2**
  - https://arm-software.github.io/CMSIS_5/RTOS2/html/rtos_api2.html