

# Elementos de sincronización en el RTOS

# Semáforos

# Coordinación entre Tareas y/o Interrupciones

- Existen distintas herramientas para coordinar las tareas entre sí, y las interrupciones con las tareas; la provisión de estos mecanismos es una de las razones para usar el OS
- Veamos la más sencilla: los semáforos, una herramienta de sincronización

## semáforo

Del gr. σῆμα *sêma* 'señal' y -foro.

1. m. Aparato eléctrico de señales luminosas para regular la circulación.
2. m. Telégrafo óptico de las costas, para comunicarse con los buques por medio de señales.
3. m. Cualquier sistema de señales ópticas. *Semáforo de banderas.*



# Semáforo

- Puede pensarse como una variable que se puede decrementar con una función **Take, Recibir o Adquirir** o incrementarse con una **Give, Dar, Soltar, Release**, y que **nunca puede valer menos de cero**
- Alternativamente (explicación de R. Barry) puede pensarse como una Queue de la cual no nos interesa los valores que almacena, sólo guardamos y recuperamos valores como forma de comunicación

```
var = 0;

funcion tomarSemaforo
  SI var>0
    var = var -1
  SI NO
    bloquear tarea
  FIN SI
fin funcion

funcion darSemaforo
  var = var + 1
fin funcion
```

# Semáforo

- La utilidad consiste en
  - Contar eventos: damos el semáforo cada vez que ocurre un evento de interés (e.g. en cada conversión del ADC) y luego vemos su valor
  - Administrar recursos: Es similar a contar eventos, por ejemplo en protocolos de comunicación complejos donde puede haber varios “clientes”, pero no infinitos, puede haber un semáforo que sólo pueda tomarse N veces
  - ¿Y qué lo diferencia de una variable común? Que es un elemento de sincronización con **operaciones atómicas**: no tendremos problemas de concurrencia en el acceso
- El semáforo binario
  - Es una herramienta sencilla para coordinar tareas, es un semáforo que **puede contar sólo hasta 1**
  - Se usa para permitir a una tarea avanzar a partir de cierto punto si se cumple un cierto evento. En FreeRTOS se distingue de los semáforos comunes por nombre

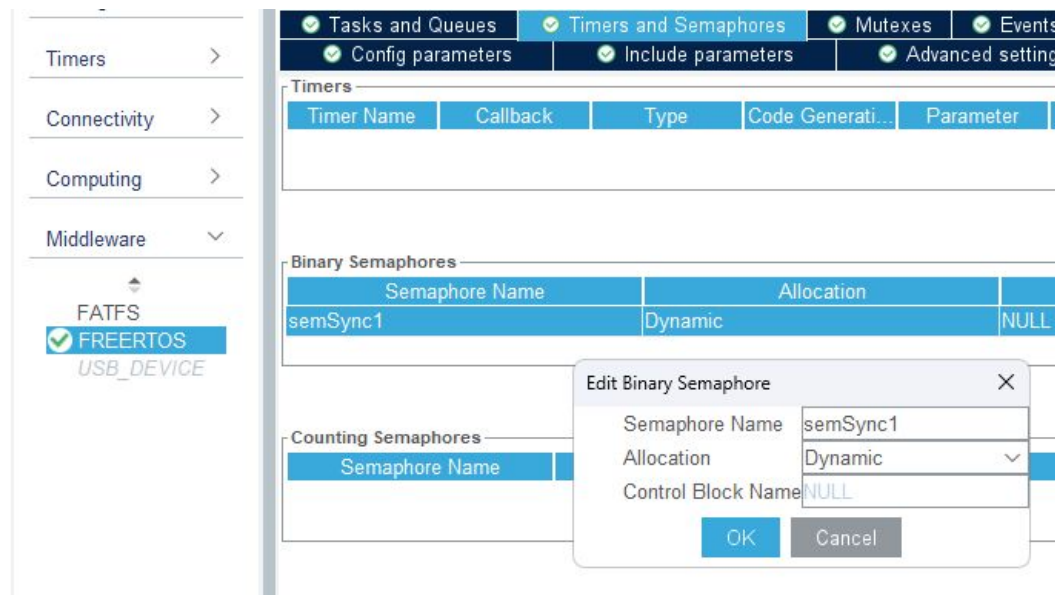
```
var = 0;
```

```
funcion tomarSemaforoBin  
    SI var>0  
        var = var -1  
    SI NO  
        bloquear tarea  
    FIN SI  
fin funcion
```

```
funcion darSemaforoBin  
    var = 1  
fin funcion
```

# Ejemplo de sincronización

- Primero, se crea un semáforo binario en el IDE



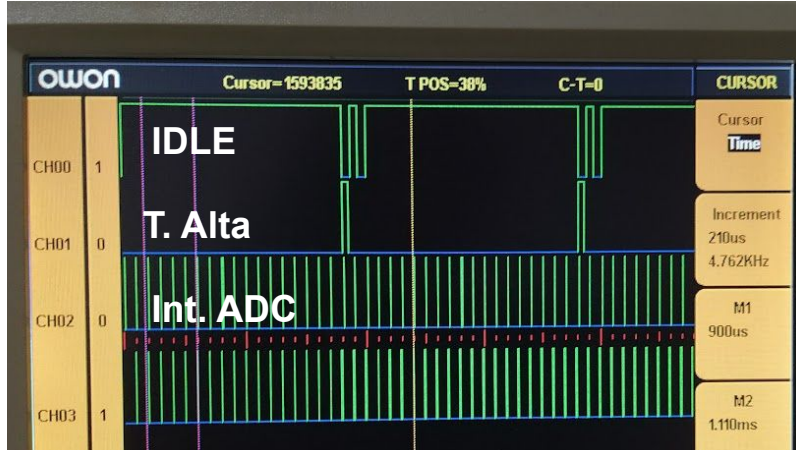
# Ejemplo de sincronización

- Y lo usamos para señalar, por ejemplo, cuando se han leído 10 valores

```
/* USER CODE END Header_entryTaskAlta */
void entryTaskAlta(void *argument)
{
    /* USER CODE BEGIN entryTaskAlta */
    vTaskSetApplicationTaskTag( NULL, (void*) TAG_TASK_ALTA);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_TIM_Base_Start_IT(&htim2);
    count = 0;
    for (;;) {
        /*La tarea se bloqueará a partir de la próxima línea*/
        osSemaphoreAcquire(semSync1Handle, osWaitForever);
        /* Cuando se libere, reseteará la variable */
        count = 0;
        osDelay(1);
    }
    /* USER CODE END entryTaskAlta */
}
```

```
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)
{
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_6,
    GPIO_PIN_SET);
    val = HAL_ADC_GetValue (&hadc1);
    count++;
    if (count== 10){
        osSemaphoreRelease (semSync1Handle);
    }
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_6,
    GPIO_PIN_RESET);
}
```

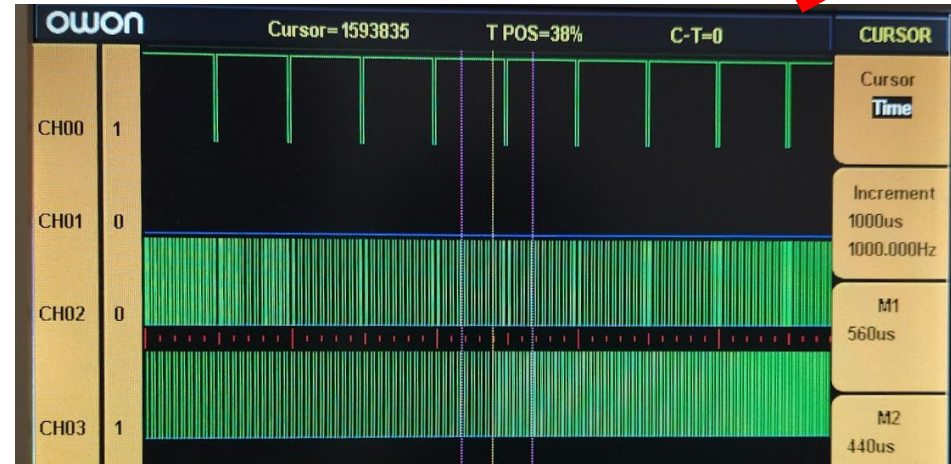
# Ejemplo de sincronización



↑ Cada 10 valores se desbloquea la tarea

Al no dar el semáforo, la tarea nunca puede tomarlo →

```
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef *hadc) {
    val = HAL_ADC_GetValue (&hadc1);
    count++;
    if(count== 10){
        //osSemaphoreRelease(semSync1Handle);
    }
}
```





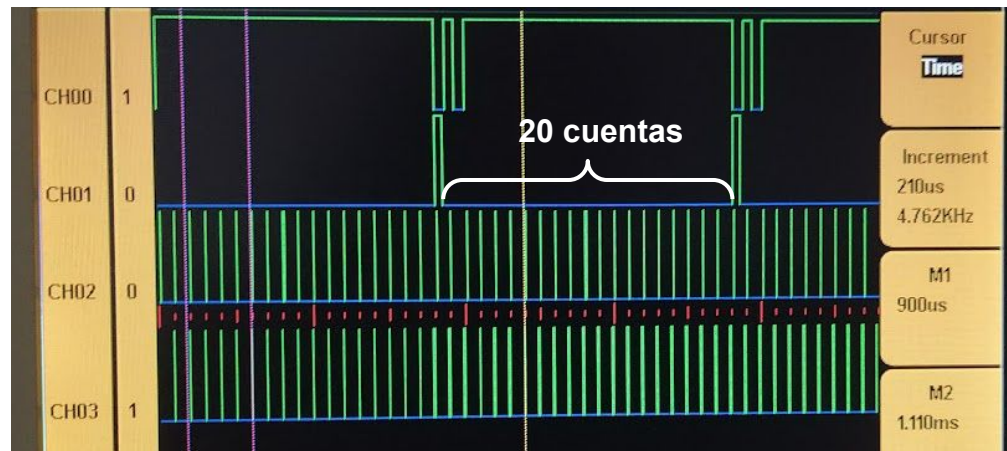
## ( Paréntesis: Funciones FromISR )

- FreeRTOS tiene una API separada para ser llamada desde las rutinas de interrupción
- Nunca deben llamarse las funciones que no son “seguras para ISR” (i.e. que terminan en FromISR) desde una interrupción
- Pero todos los elementos en general cuentan con sus APIs compatibles con interrupciones
- CMSIS v2 nos esconde esto y lo resuelve

```
osStatus_t osSemaphoreRelease (osSemaphoreId_t semaphore_id) {  
    SemaphoreHandle_t hSemaphore = (SemaphoreHandle_t) semaphore_id;  
    ...  
    if (hSemaphore == NULL) {  
        stat = osErrorParameter;  
    }  
    else if (IS_IRQ()) {  
        ...  
        xSemaphoreGiveFromISR (hSemaphore, &yield)  
    }  
    else {  
        if (xSemaphoreGive (hSemaphore) != pdPASS) {  
            stat = osErrorResource;  
        }  
    }  
  
    return (stat);  
}
```

## Ejemplo de sincronización

- Cada vez que `count==10` se “libera” la ejecución de la tarea
- Pero esto no significa que la tarea se ejecute
- Sólo significa que cuando el scheduler la venga a buscar, la encontrará en estado READY y ya no bloqueada



| Live Expressions                       |         |                               |
|--|---------|-------------------------------|
| Expressions (x)= Variables Breakpoints |         |                               |
| Expression                             | Type    | Value                         |
| prom                                   |         | Failed to evaluate expression |
| std                                    |         | Failed to evaluate expression |
| (x)= val                               | int16_t | 1026                          |
| (x)= count                             | int     | 14                            |
| + Add new expression                   |         |                               |

# Ejemplo de sincronización

- Puede observarse con un cambio menor en la visualización
- En este caso, si tenemos un buffer de 10 muestras, se desbordaría... ¿podemos “invocar” a la tarea?

```
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc) {  
    //HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);  
    val = HAL_ADC_GetValue (&hadc1);  
    count++;  
    if(count== 10){  
        HAL_GPIO_WritePin (GPIOB, GPIO_PIN_6, GPIO_PIN_SET);  
        osSemaphoreRelease (semSync1Handle);  
    }  
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);  
}
```



## portYIELD\_FROM\_ISR()

- taskYIELD() se puede invocar dentro de una tarea para pedir un cambio de contexto a otra de mayor prioridad
- **portYIELD\_FROM\_ISR()** es una versión para **usar desde interrupciones** y permitirá que al terminar una interrupción se pida al scheduler un cambio de contexto
- El scheduler realizará el cambio **si hay una tarea de alta prioridad en estado Ready**
- La función toma un parámetro que por convención dentro de FreeRTOS se nombra **xHigherPriorityTaskWoken**




```
BaseType_t xHigherPriorityTaskWoken;  
xHigherPriorityTaskWoken = pdFALSE;  
xHigherPriorityTaskWoken = pdTRUE;
```

- Si el parámetro xHigherPriorityTaskWoken pasado a portYIELD\_FROM\_ISR() es pdFALSE, entonces la función (es un macro en realidad) no pide ningún cambio de contexto. **Si es pdTRUE se pide un cambio de contexto** y puede cambiar la tarea RUNNING
- portYIELD\_FROM\_ISR() Debe llamarse al final de la rutina de interrupción

## xHigherPriorityTaskWoken

- Las funciones de la API de FreeRTOS usan este parámetro y se le puede pasar con el valor pdTRUE para indicar que luego de hacer lo que hagan (e.g. entregar un semáforo) se pide un cambio de contexto
- Pero CMSIS v2 al manipular los llamados en una API que sirve tanto para tareas como para ISRs “toquetea” este parámetro
- Podemos usar el macro PortYield directamente

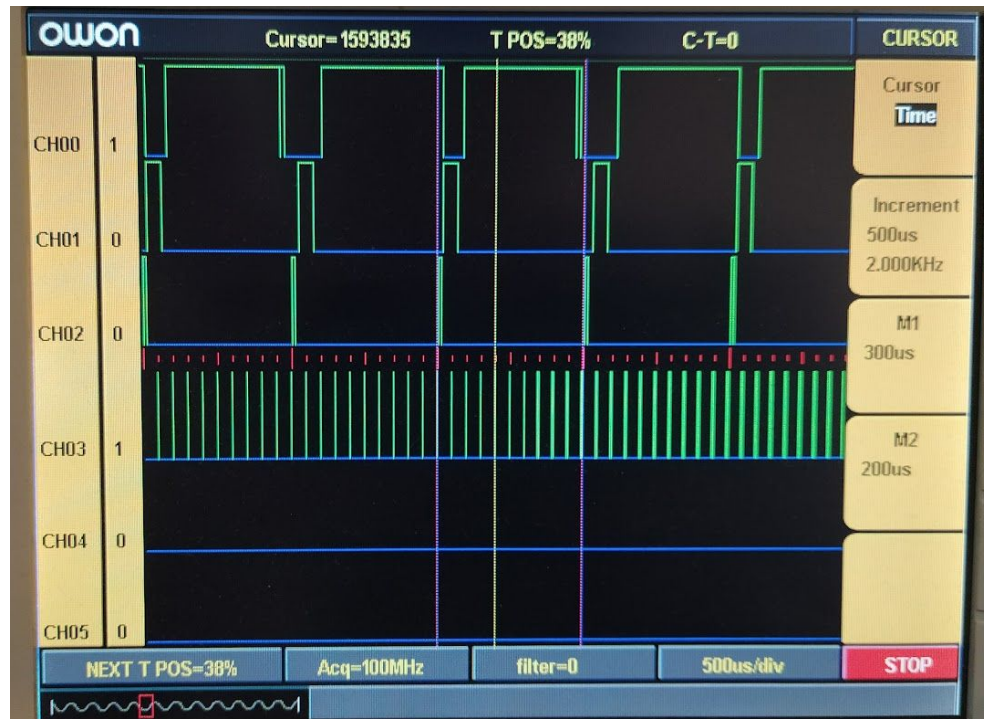
# xHigherPriorityTaskWoken

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {  
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;   
    val = HAL_ADC_GetValue(&hadc1);  
    count++;  
    if(count== 10){  
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);  
        xHigherPriorityTaskWoken = pdTRUE;   
        osSemaphoreRelease(semSynclHandle);  
    }  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);  
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );   
}
```

# Ejemplo de sincronización



antes y después

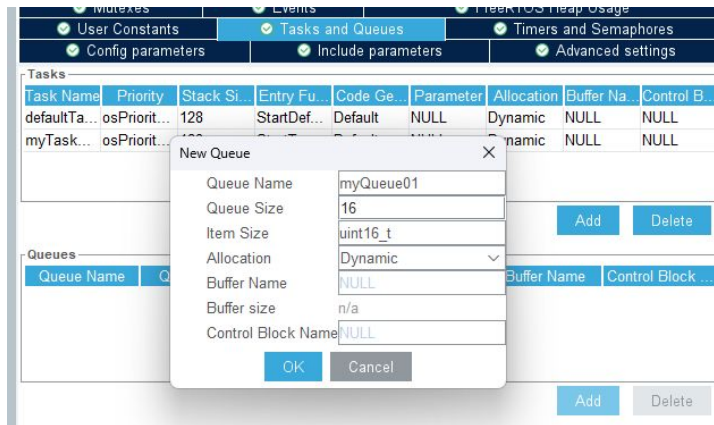


# Colas



# Colas en FreeRTOS

- Una cola o Queue puede contener un número finito de ítems de tamaño fijo que se configuran al momento de su creación.
- El tamaño máximo que puede contener es su largo o tamaño (length, size)



- Son buffers FIFO y al escribir un dato el mismo se copia al buffer que mantiene la cola. Al leer, el dato se borra.

# Colas en FreeRTOS

- La API de CMSIS V2 nos muestra las operaciones que pueden hacerse:
  - **osMessageQueueDelete** : Delete a Message Queue object.
  - **osMessageQueueGet** : Get a Message from a Queue or timeout if Queue is empty.
  - **osMessageQueueGetCapacity** : Get maximum number of messages in a Message Queue.
  - **osMessageQueueGetCount** : Get number of queued messages in a Message Queue.
  - **osMessageQueueGetMsgSize** : Get maximum message size in a Message Queue.
  - **osMessageQueueGetName** : Get name of a Message Queue object.
  - **osMessageQueueGetSpace** : Get number of available slots for messages in a Message Queue.
  - **osMessageQueueNew** : Create and Initialize a Message Queue object.
  - **osMessageQueuePut** : Put a Message into a Queue or timeout if Queue is full.
  - **osMessageQueueReset** : Reset a Message Queue to initial empty state.

# Lo usamos en el ejemplo del muestreo

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {

    int16_t val;
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    val = HAL_ADC_GetValue(&hadc1);
    osStatus_t res = osMessageQueuePut(queueDatosHandle, &val, 0, 0);
    if(res != osOK){
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    }
    count++;
    if(count== NS){
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
        count = 0;
        xHigherPriorityTaskWoken = pdTRUE;
        osSemaphoreRelease(semSync1Handle);
    }
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

# Procesamiento de media y std

```
void entryTaskAlta(void *argument){ ...
for (;;) {
    osSemaphoreAcquire (semSynclHandle, osWaitForever);    // Sincronización
    acc=0;          // Reset de variables
    istd = 0;       //
    for(i=0; i<N; i++){    // Cálculo de promedio y desviación estándar
        res = osMessageQueueGet (queueDatosHandle, &val, 0, 0); // Recepción de queue
        if(res == osOK ) acc += ( float)val;                    // Si no hubo errores, sumar
        else HAL_GPIO_WritePin (GPIOC, GPIO_PIN_13, GPIO_PIN_SET); // Si los hubo, prender luz
        dif = ( int)val-prom;    // xi- prom
        istd += dif*dif;         // sum [ (xi-prom)^2 ]
    }
    acc = acc/N;
    prom = acc;                  // Actualizacion var global con promedio
    istd = istd/N;
    std = sqrt(istd);            // Actualización var global con desv est
}}
```

# Completamos el ejemplo



# Procesamiento de interrupciones diferido

- En este ejemplo se repartió la carga de forma “secuencial” para demostrar el uso de semáforos y colas
- Si hay otras tareas para realizar en el OS, se pueden configurar para ejecutar concurrentemente
- Tenemos un factor de utilización del 80%



# Notificaciones

# Task Notifications

- Hasta ahora los elementos de sincronización son **externos** a las tareas
  - Queue
  - Mutex
  - Semáforo
- Las notificaciones a tareas consisten en un campo opcional en el TBC de las tareas y permiten escribir 1 flag y 1 valor usando una API de FreeRTOS
- A diferencia de los otros elementos, son propias de cada tarea, esto dicta la lógica de funcionamiento (por ejemplo, no se puede notificar **a una** ISR, pero si **desde una** ISR)

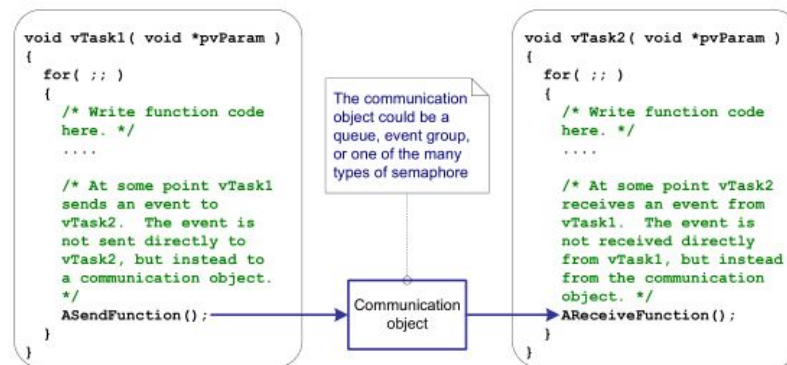
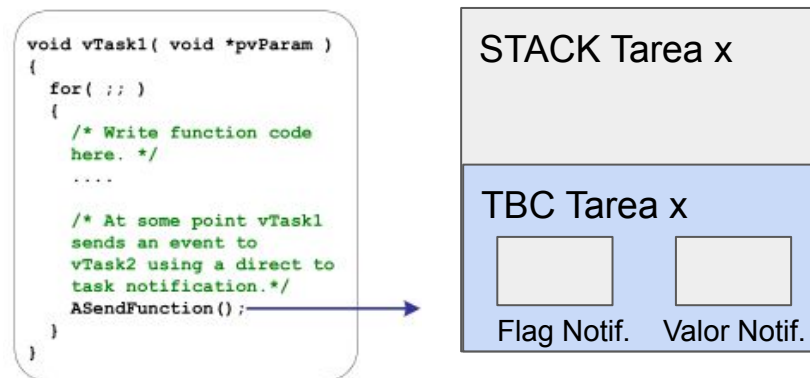


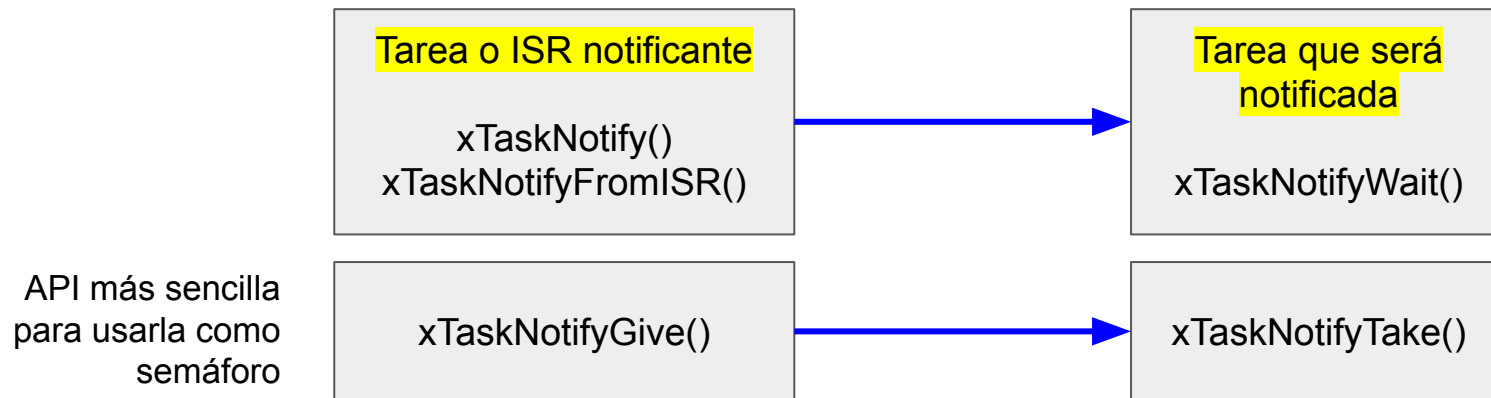
Figure 76 A communication object being used to send an event from one task to another





# Task Notifications

When `configUSE_TASK_NOTIFICATIONS` is set to 1, each task has a 'Notification State', which can be either 'Pending' or 'Not-Pending', and a 'Notification Value', which is a 32-bit unsigned integer. When a task receives a notification, its notification state is set to pending. When a task reads its notification value, its notification state is set to not-pending.



# Task Notifications

- La API simplificada usa los campos de notificación para emular un semáforo:
  - Setea el campo de notificación
  - Incrementa (en el caso de Give) en uno la variable

---

```
BaseType_t xTaskNotifyGive( TaskHandle_t xTaskToNotify );
```

---

**Listing 145. The xTaskNotifyGive() API function prototype**

---

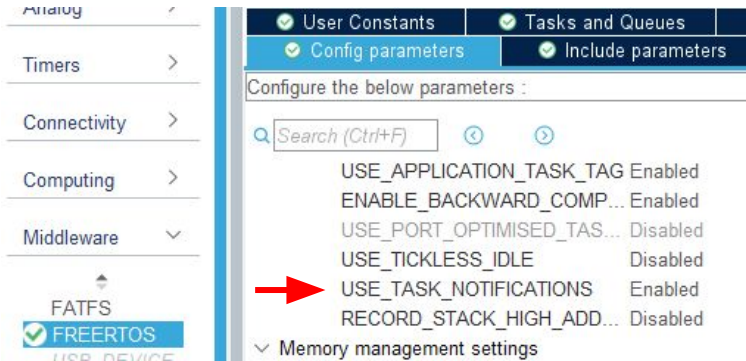
```
void vTaskNotifyGiveFromISR( TaskHandle_t xTaskToNotify,  
                             BaseType_t *pxHigherPriorityTaskWoken );
```

---

**Listing 146. The vTaskNotifyGiveFromISR() API function prototype**

---

# Ejemplo de uso.

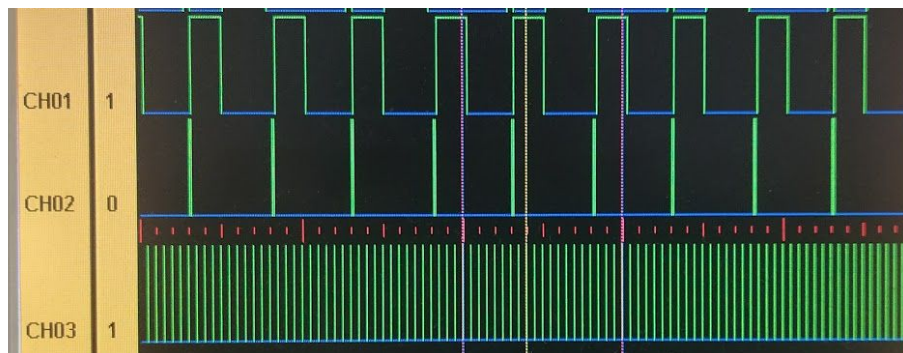


```
for (;;) {
    // uint32_t ulTaskNotifyTake(
    // BaseType_t xClearCountOnExit,
    // TickType_t xTicksToWait);
    ulTaskNotifyTake(pdFALSE, portMAX_DELAY);
    acc=0;
    for(i=0; i<N; i++){ ... }
    acc = acc/N;
    prom = acc;
}
```

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    val = HAL_ADC_GetValue(&hadc1);
    osStatus_t res = osMessageQueuePut(queueDatosHandle, &val, 0, 0);
    if(res != osOK) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    count++;
    if(count== NS){
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
        count = 0;
        vTaskNotifyGiveFromISR(TaskAltaHandle, &xHigherPriorityTaskWoken);
        xHigherPriorityTaskWoken = pdTRUE;
    }
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

(Reemplazamos el elemento de sincronización del ejemplo anterior)

## Ejemplo de uso



`vTaskNotifyGiveFromISR` toma por referencia el valor `xHigherPriorityTaskWoken`. Si es `pdTRUE`, significa que la tarea notificada pasará al estado `RUNNING`

```
// uint32_t ulTaskNotifyTake(  
// BaseType_t xClearCountOnExit,  
// TickType_t xTicksToWait);  
ulTaskNotifyTake(pdFALSE, portMAX_DELAY);  
  
vTaskNotifyGiveFromISR(TaskAltaHandle,  
                        &xHigherPriorityTaskWoken);
```

`ulTaskNotifyTake` decrementa en 1 el valor de notificación, o lo hace 0 si el 1er parámetro es `pdTRUE`. Devuelve como `uint32_t` el valor previo a su borrado o decremento, si se quiere usar como semáforo con conteo.

# xTaskNotifyFromISR()

Da un acceso “completo” a la funcionalidad de las notificaciones. Se puede usar para

- Incrementar el valor de notificación (usándola igual que la versión Give)
- Setear un valor particular para señalizar mensajes. Esta funcionalidad está pensada para usarla seteando bits del valor de notificación.
- Sobreescribir el valor de notificación sólo si la tarea receptora lo ha leído desde su última actualización (se comporta como una cola de 1 sólo elemento).
- Sobreescribir el valor de notificación sin importar si se ha leído o no

```
BaseType_t xTaskNotify( TaskHandle_t xTaskToNotify,
                        uint32_t ulValue,
                        eNotifyAction eAction );

BaseType_t xTaskNotifyFromISR( TaskHandle_t xTaskToNotify,
                              uint32_t ulValue,
                              eNotifyAction eAction,
                              BaseType_t *pxHigherPriorityTaskWoken );
```

|                           |  |
|---------------------------|--|
| eNoAction                 | No hace nada. Equivalente a semáforo binario   |
| eSetBits                  | Se hace un OR binario entre el valor de notificación que se pasa (ulValue) y el actual |
| eIncrement                | Se incrementa en 1 el valor de notificación actual (ulValue no se usa)                 |
| eSetValueWithoutOverwrite | Si la notificación estaba en estado pendiente no se sobreescribe y devuelve pdFAIL     |
| eSetValueWithOverwrite    | Se sobreescribe el valor con ulValue sin importar el estado de notificación            |

# xTaskNotifyWait()

Espera con un timeout opcional a que se le envíe una notificación. Sus argumentos permiten:

- **ulBitsToClearOnEntry:** Borrar los bits del valor de notificación al entrar a la función (si no tenía una notificación pendiente). Es decir que para borrar a 0 hay que pasarle 0xFFFFFFFF
- **ulBitsToClearOnExit:** Borrar los bits del valor de notificación al salir de la función, pero previamente se almacena el valor en la variable apuntada por el 3er argumento
- **pulNotificationValue:** Recupera el valor de notificación antes de ser borrado OnExit
- **xTicksToWait:** Timeout en ticks. Puede usarse `pdMS_TO_TICKS()` para introducir el valor en ms o `portMAX_DELAY` para que la espera sea “infinita”

Y su valor de retorno puede ser

- **pdTRUE:** Se recibió la notificación esperada ya sea luego de haber estado esperando en estado bloqueado o previo al llamado de la función
- **pdFALSE:** La función retornó sin haber recibido la notificación (ocurrió el timeout)

```
BaseType_t xTaskNotifyWait( uint32_t ulBitsToClearOnEntry,  
                             uint32_t ulBitsToClearOnExit,  
                             uint32_t *pulNotificationValue,  
                             TickType_t xTicksToWait );
```

## Ejemplo modo “semáforo binario”

```
void entryTaskNormal(void *argument)
{
    /* USER CODE BEGIN entryTaskNormal */
    vTaskSetApplicationTaskTag( NULL, (void*) TAG_TASK_NORMAL);
    uint8_t data;
    /* Infinite loop */
    for (;;) {
        HAL_UART_Receive_IT(&huart1, &data, 1);
        xTaskNotifyWait(0U, 0U, NULL, portMAX_DELAY);
        HAL_UART_Transmit(&huart1, "ACK\n\r", 6, 100);
    }
    /* USER CODE END entryTaskNormal */
}
```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    BaseType_t pxHigherPriorityTaskWoken = pdFALSE;
    xTaskNotifyFromISR(TaskNormalHandle, 0U, eNoAction, &pxHigherPriorityTaskWoken);
}
```

# Ejemplo con pasaje de datos

```
void entryTaskNormal(void *argument)
{
    ...
    for (;;) {
        if(recibido==1){
            HAL_UART_Receive_IT(&huart1, &data, 1);
            recibido = 0;
        }
        xTaskNotifyWait(0U, 0U, &val_notif, portMAX_DELAY);
        if(val_notif == TX_ACK){
            recibido = 1;
            HAL_UART_Transmit(&huart1, "ACK\n\r", 6, 100);
        }
        else if(val_notif == TX_VAL){
            numchar = sprintf(num,"%d\n\r",prom_int);
            HAL_UART_Transmit(&huart1, num, numchar+1, 100);
        }
    }
    /* USER CODE END entryTaskNormal */
}
```

```
#define TX_ACK 1U
#define TX_VAL 2U
```

```
void entryTaskAlta(void *argument)
{
    ...
    for (;;) {
        ulTaskNotifyTake(pdFALSE, portMAX_DELAY);
        acc=0;
        for(i=0; i<N; i++){...}
        acc = acc/N;
        prom_int = (int)acc;
        xTaskNotify(TaskNormalHandle, TX_VAL,
                    eSetValueWithOverwrite);
    }
}
```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    BaseType_t pxHigherPriorityTaskWoken = pdFALSE;
    xTaskNotifyFromISR(TaskNormalHandle, TX_ACK,
                        eSetValueWithOverwrite,&pxHigherPriorityTaskWoken);
}
```

Tera Term -

File Edit Setu

```
2018
2018
2017
2018
2017
2018
2017
2017
2018
2018
2017
2017
2017
2017
ACK
2017
2017
2017
2017
2017
2017
```