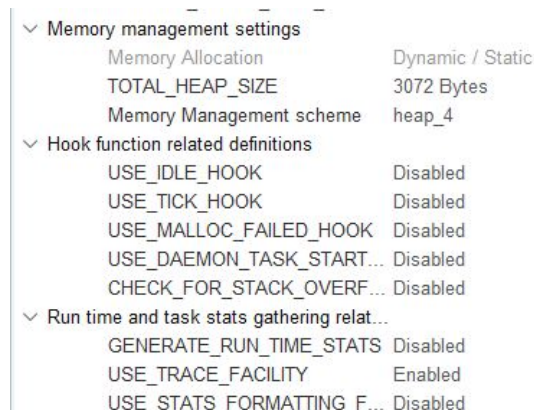
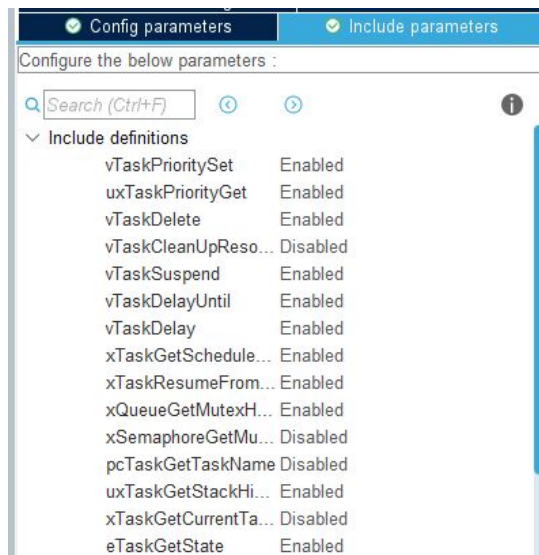
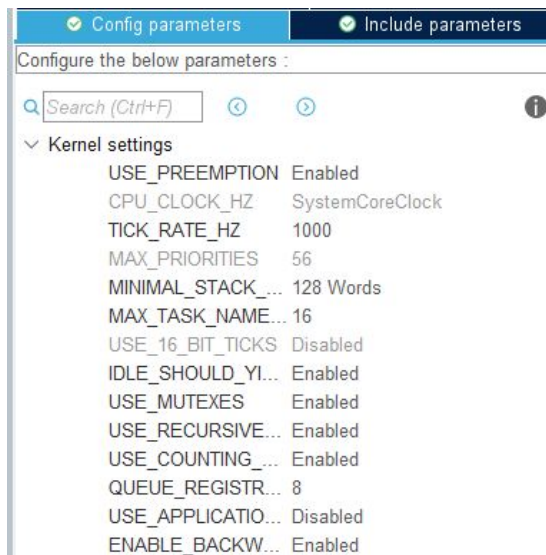
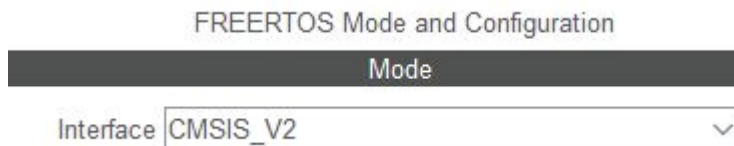


El sistema operativo y las tareas/threads

Tareas - Interrupciones

Instalando FreeRtos con capa CMSIS OS V2 desde Cube IDE



Menú para el agregado de Tareas

✓ User Constants	✓ Tasks and Queues	✓ Timers and Semaphores
✓ Config parameters	✓ Include parameters	✓ Advanced settings

Tasks

Task N...	Priority	Stack ...	Entry F...	Code G...	Param...	Allocati...	Buffer ...	Control...
default...	osPrior...	128	StartD...	Default	NULL	Dynamic	NULL	NULL

Default task, mainly in charge of middlewares init (when some are selected).
Can be modified but not removed!
Double-click to edit and modify.

Add

Delete

```

275 /* USER CODE BEGIN Header_comenzarTareaParpadeo */
276 /**
277  * @brief Function implementing the tareaParpadeo thread.
278  * @param argument: Not used
279  * @retval None
280  */
281 /* USER CODE END Header_comenzarTareaParpadeo */
282 void comenzarTareaParpadeo(void *argument) {
283     /* USER CODE BEGIN comenzarTareaParpadeo */
284     /* Infinite loop */
285     for (;;) {
286         osDelay(500);
287         HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
288     }
289     /* USER CODE END comenzarTareaParpadeo */
290 }
291

```

User Constants		Tasks and Queues		Timers and Semaphores	
Config parameters		Include parameters		Advanced settings	
Tasks					
Task N...	Priority	Stack ...	Entry F...	Code G...	Param...
default...	osPrior...	128	StartD...	Default	NULL
<div> <div>Task Name</div> <div>tareaParpadeo</div> </div> <div> <div>Priority</div> <div>osPriorityLow</div> </div> <div> <div>Stack Size (Words)</div> <div>128</div> </div> <div> <div>Entry Function</div> <div>comenzarTareaParpadeo</div> </div> <div> <div>Code Generation Option</div> <div>Default</div> </div> <div> <div>Parameter</div> <div>NULL</div> </div> <div> <div>Allocation</div> <div>Dynamic</div> </div> <div> <div>Buffer Name</div> <div>NULL</div> </div> <div> <div>Control Block Name</div> <div>NULL</div> </div>					
<div> <div>Add</div> <div>Delete</div> </div>					
<div> <div>OK</div> <div>Cancel</div> </div>					

Las tareas/threads se generan a partir de funciones

- Recordemos que las tareas son funciones de C con un bucle infinito (nunca retornan)
- Cada tarea tiene un TBC y un stack propio asignado en el Heap que reservó el sistema operativo
- En ese stack puede guardar sus variables locales y, si llama a funciones, allí hará el stacking

Todo lo que declaremos aquí va a durar el tiempo de ejecución de la tarea y va a ser local a esa tarea.
Podríamos declarar más de una tarea aprovechando la misma función (sólo compartirán variables static)

Verán, como cualquier función, las variables globales y podrán compartir información con otras Tareas a través de ellas

Aquí implementaremos el código que la tarea debe realizar

Y nunca deberíamos llegar a este punto, sin BORRAR la tarea

```
void TareaA(void* args) {  
  
    for (;;) {  
  
    }  
  
}
```

Estados de las tareas

- Mejoremos el diagrama de los estados de las tareas que vimos anteriormente
- En un sistema con un sólo núcleo una sólo tarea por vez puede estar corriendo
- Esa será la tarea con el estado “**RUNNING**”
- Cuando no está corriendo, una tarea puede estar en alguno de los siguientes estados:
 - **READY** Está lista (y quiere) para que el scheduler la ponga a correr.
 - **BLOCKED** Si por algún motivo debe esperar un evento y el scheduler no debe tomarla en cuenta para hasta que ese evento ocurra.
 - **SUSPENDED** (o Inactive en CMSIS OS) Suspendida, hasta que no se la reanude no volverá a necesitar ejecutarse (i.e. sale de las tareas analizadas por el scheduler).

```
typedef enum {  
    osThreadInactive      = 0, ///  
    osThreadReady         = 1, ///  
    osThreadRunning       = 2, ///  
    osThreadBlocked       = 3, ///  
    osThreadTerminated    = 4, ///  
    osThreadError         = -1, ///  
} osThreadState_t;
```

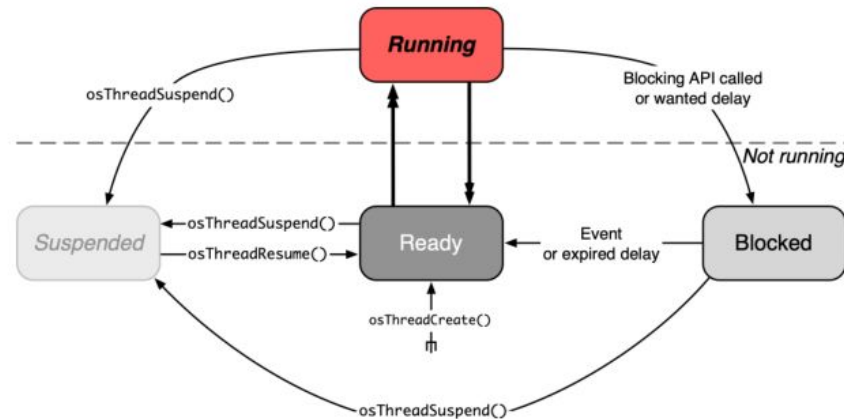


Figure 23.7: The possible states of a thread in FreeRTOS

Estrategia de Scheduling

- El RTOS configurado con CubeMX (o el FreeRTOS instalado “como viene”) tienen por default la estrategia **Preemptive con prioridades y Time Slice**
- Cada Tarea tiene una **prioridad fija**, asignada al crearla. El OS no puede cambiar esa prioridad (pero puede modificarse usando `osThreadSetPriority()`)
- El scheduler **cambiará la tarea cuando una de mayor prioridad pase al estado “READY”**, aunque la que se esté ejecutando no pase voluntariamente al estado Blocked o Suspended
- Entre tareas de igual prioridad se usará el time slice. Cuando se pasa la “tajada” el scheduler va a seleccionar la próxima tarea de una lista de READYs y le va a asignar nuevamente una tajada. Si no hay ninguna, pone a correr la tarea IDLE.
- Por defecto, la time slice y el tick del RTOS son iguales y de 1ms
- Cuando una tarea quiere ceder la ejecución puede usar la función `osTaskYield()` o generar un delay.

<code>configUSE_PREEMPTION</code>	<code>configUSE_TIME_SLICING</code>	Scheduling algorithm
1	1 or undefined	<i>Prioritized preemptive scheduling with time slicing</i>
1	0	<i>Prioritized preemptive scheduling without time slicing</i>
0	any value	<i>Cooperative scheduling</i>

Creación de tareas

```
/// Attributes structure for thread.
typedef struct {
    const char                *name;    ///< name of the thread
    void                    *cb_mem;    ///< memory for control block
    uint32_t                cb_size;    ///< size of provided memory for control block
    void                    *stack_mem; ///< memory for stack
    uint32_t                stack_size; ///< size of stack
    osPriority_t             priority;   ///< initial thread priority (default: osPriorityNormal)
    TZ_ModuleId_t           tz_module;  ///< TrustZone module identifier
    uint32_t                reserved;   ///< reserved (must be 0)
} osThreadAttr_t;

osThreadId_t osThreadNew (osThreadFunc_t func, void *argument, const osThreadAttr_t *attr) {
    prio = (UBaseType_t)osPriorityNormal;
    name = attr->name;
    prio = (UBaseType_t)attr->priority;
    stack = attr->stack_size / sizeof(StackType_t);
    xTaskCreate ((TaskFunction_t)func, name, (uint16_t)stack, argument, prio, &hTask)
}
```

“Resumen” de la
función osThreadNew

Argumentos de las tareas

- El argumento es puntero a void para que se pueda castear cualquier cosa en ese formato y pasárselo a la tarea.
- Un ejemplo: dos tareas para hacer parpadear **distintos leds a distinta frecuencia** a partir de **la misma función**

```
/* USER CODE BEGIN PV */  
struct configParpadeo_t {  
    GPIO_TypeDef *puerto;  
    uint32_t pin;  
    uint32_t periodo;  
};  
struct configParpadeo_t conf1 =  
    {GPIOA, GPIO_PIN_5, 500};  
struct configParpadeo_t conf2 =  
    {GPIOA, GPIO_PIN_6, 1500};  
/* USER CODE END PV */
```

Task	
Task Name	tareaParpadeo
Priority	osPriorityLow
Stack Size (Words)	128
Entry Function	entryTareaParpadeo
Code Generation Option	Default
Parameter	&conf1
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL
<div>OK Cancel</div>	

Task	
Task Name	tareaParpadeo2
Priority	osPriorityLow
Stack Size (Words)	128
Entry Function	entryTareaParpadeo
Code Generation Option	Default
Parameter	&conf2
Allocation	Dynamic
Buffer Name	NULL
Control Block Name	NULL
<div>OK Cancel</div>	

Argumentos de las tareas

- El argumento es puntero a void para que se pueda castear cualquier cosa en ese formato y pasárselo a la tarea.
- Un ejemplo: dos tareas para hacer parpadear **distintos leds a distinta frecuencia** a partir de **la misma función**

```
/* creation of tareaParpadeo */
tareaParpadeoHandle = osThreadNew(
    entryTareaParpadeo,
    (void*) &conf1,
    &tareaParpadeo_attributes    );

/* creation of tareaParpadeo2 */
tareaParpadeo2Handle = osThreadNew(
    entryTareaParpadeo,
    (void*) &conf2,
    &tareaParpadeo2_attributes    );
```

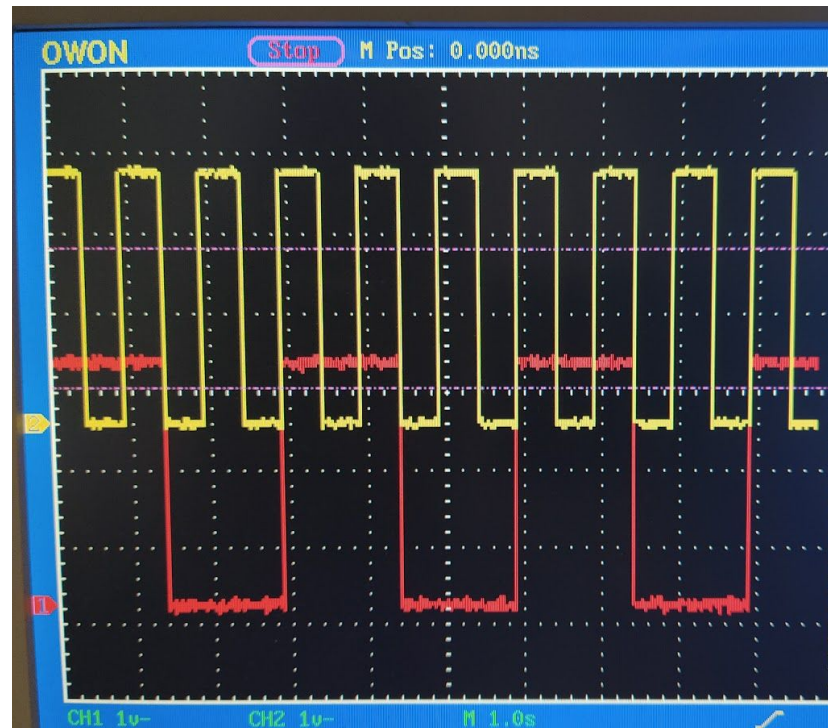
```
void entryTareaParpadeo(void *argument)
{
    /* USER CODE BEGIN entryTareaParpadeo */
    struct configParpadeo_t conf =
        *((struct configParpadeo_t*) argument);
    /* Infinite loop */
    for(;;)
    {
        HAL_GPIO_TogglePin(conf.puerto, conf.pin);
        osDelay(pdMS_TO_TICKS(conf.periodo));
    }
    /* USER CODE END entryTareaParpadeo */
}
```

Argumentos de las tareas

```
/* Variables globales */
struct configParpadeo_t conf1 =
{GPIOA, GPIO_PIN_5, 500};
struct configParpadeo_t conf2 =
{GPIOA, GPIO_PIN_6, 1500};

/* Instanciación de tareas en el main()*/
osThreadNew(entryTareaParpadeo, (void*) &conf1, ...);
osThreadNew(entryTareaParpadeo, (void*) &conf2, ...);

/* Función de las tareas */
void entryTareaParpadeo(void *argument){
    struct configParpadeo_t conf = *argument;
    for(;;){
        HAL_GPIO_TogglePin(conf.puerto, conf.pin);
        osDelay(pdMS_TO_TICKS(conf.periodo));
    }
}
```



Análisis del Stack

Explorando el stack

- Exploremos el stack de las tareas del ejemplo anterior
- Le dimos una “profundidad” (depth) de 128 palabras o $128 \times 4 = 512$ bytes
- ¿Está bien esto? Para saberlo deberíamos averiguar
 - El espacio que ocupan todas sus variables locales
 - Todas las funciones que llama (y aquellas que son llamadas por ellas) (call-graph)
 - El stack que necesitan todas las funciones en ese call-graph
 - Y por último calcular el peor caso
- Y ahora, ¿Quién podrá ayudarnos?

Explorando el stack

- Afortunadamente, el compilador de ARM tiene toda esta información y hace las cuentas para resolverlo por nosotros

Static Stack Analyzer × Console Problems Executables Debugger Console

laboratorio_tasks_programa_2.elf - /laboratorio_tasks_programa_2/Debug - Oct 30, 2022

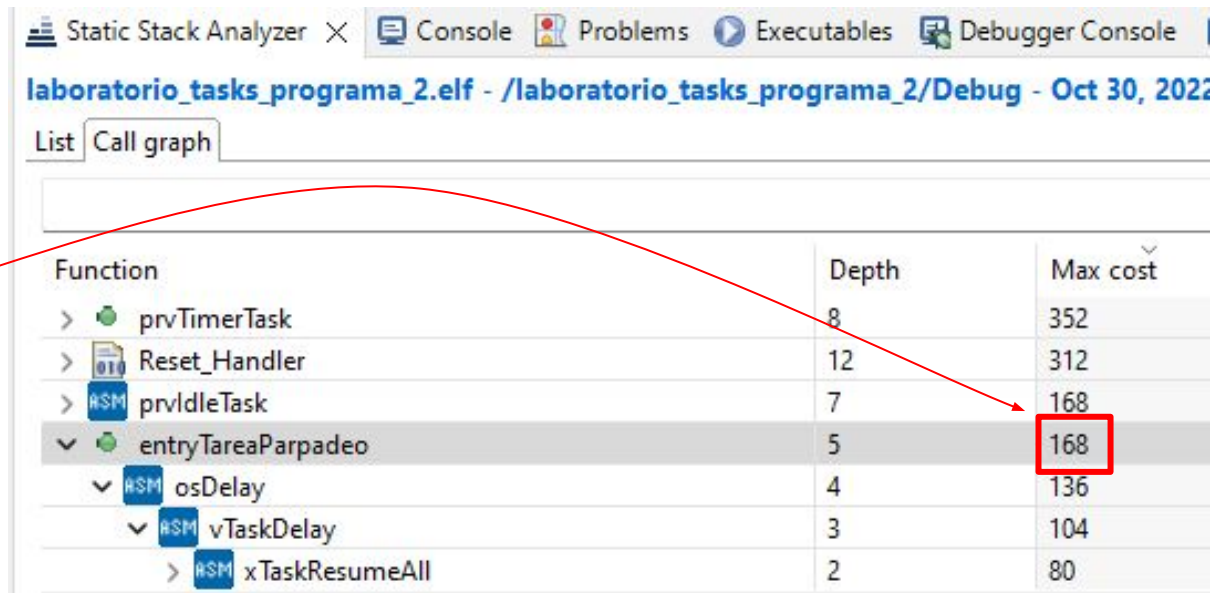
List Call graph

Function	Depth	Max cost
> prvTimerTask	8	352
> Reset_Handler	12	312
> prvIdleTask	7	168
▼ entryTareaParpadeo	5	168
▼ osDelay	4	136
▼ vTaskDelay	3	104
> xTaskResumeAll	2	80



Explorando el stack - En la compilación

- Para todas las funciones, analiza el “Call graph” que ve todos los posibles llamados de funciones y el “árbol” de llamadas que se desprende de cada una
- Devuelve el “costo máximo” que es el peor caso que logra estimar



Static Stack Analyzer × Console Problems Executables Debugger Console

laboratorio_tasks_programa_2.elf - /laboratorio_tasks_programa_2/Debug - Oct 30, 2023

List Call graph

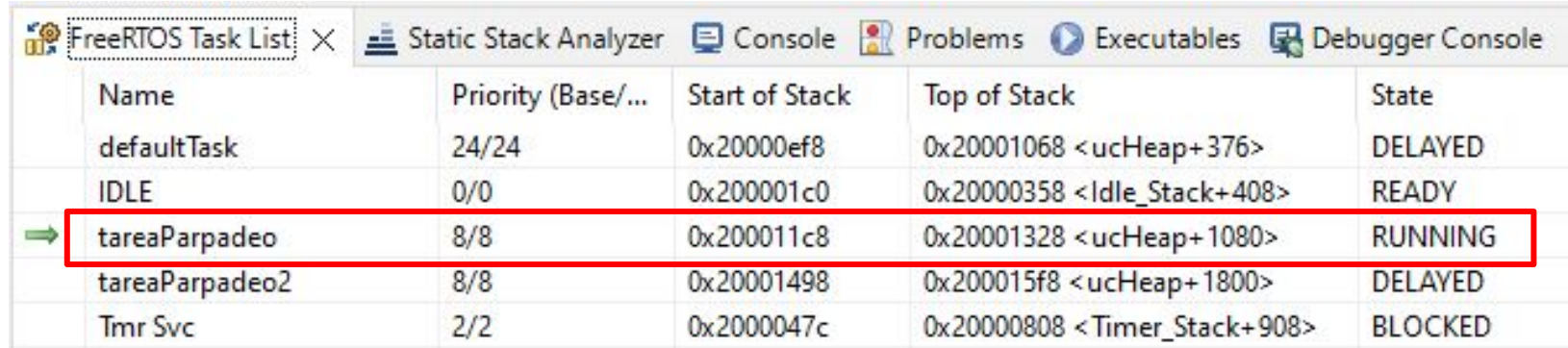
Function	Depth	Max cost
> prvTimerTask	8	352
> Reset_Handler	12	312
> prvIdleTask	7	168
✓ entryTareaParpadeo	5	168
✓ osDelay	4	136
✓ vTaskDelay	3	104
> xTaskResumeAll	2	80

Explorando el stack - En la ejecución

- Comprobemos la medición de máximo costo observando qué pasa con el stack de la tarea parpadeo durante la ejecución, deteniéndose con un breakpoint en un punto

```
277 void entryTareaParpadeo(void *argument)
278 {
279     /* USER CODE BEGIN entryTareaParpadeo */
280
281     struct configParpadeo_t conf =
282         *((struct configParpadeo_t*)argument);
283     /* Infinite loop */
284     for(;;)
285     {
286         HAL_GPIO_TogglePin(conf.puerto, conf.pin);
287         osDelay(pdMS_TO_TICKS(conf.periodo));
288     }
289     /* USER CODE END entryTareaParpadeo */
290 }
```


Explorando el stack - En la ejecución



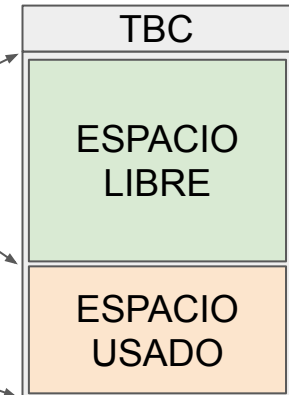
Name	Priority (Base/...	Start of Stack	Top of Stack	State
defaultTask	24/24	0x20000ef8	0x20001068 <ucHeap+376>	DELAYED
IDLE	0/0	0x200001c0	0x20000358 <Idle_Stack+408>	READY
→ tareaParpadeo	8/8	0x200011c8	0x20001328 <ucHeap+1080>	RUNNING
tareaParpadeo2	8/8	0x20001498	0x200015f8 <ucHeap+1800>	DELAYED
Tmr Svc	2/2	0x2000047c	0x20000808 <Timer_Stack+908>	BLOCKED

- Recordemos que el stack pointer es full-descending. Es decir que **arranca por la dirección más alta** y desciende hacia la más baja

START - La dir. más baja
(final del stack)

TOP - Stack pointer actual

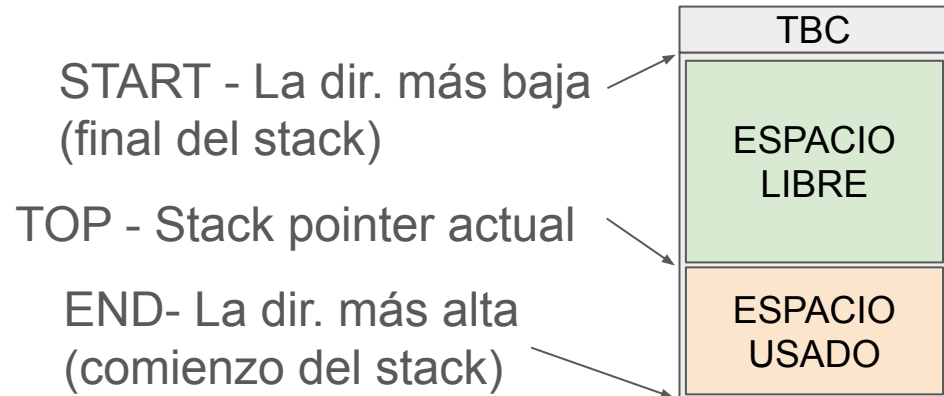
END- La dir. más alta
(comienzo del stack)



Explorando el stack - En la ejecución

FreeRTOS Task List					
Name	Priority (Base/...	Start of Stack	Top of Stack	State	
defaultTask	24/24	0x20000ef8	0x20001068 <ucHeap+376>	DELAYED	
IDLE	0/0	0x200001c0	0x20000358 <Idle_Stack+408>	READY	
→ tareaParpadeo	8/8	0x200011c8	0x20001328 <ucHeap+1080>	RUNNING	
tareaParpadeo2	8/8	0x20001498	0x200015f8 <ucHeap+1800>	DELAYED	
Tmr Svc	2/2	0x2000047c	0x20000808 <Timer_Stack+908>	BLOCKED	

tareaParpadeo			
	byte (hex)	word (dec)	byte (dec)
Start	11c8		4552
Top	1328		4904
Depth	200	128	512
End	13c8		5064
Usado	A0	40	160



Explorando el stack

- Podemos visualizar la memoria RAM directamente para examinar la tarea

tareaParpadeo			
	byte (hex)	word (dec)	byte (dec)
Start	11c8		4552
Top	1328		4904
Depth	200	128	512
End	13C8		5064
Usado	A0	40	160

Memory × FreeRTOS Task List Static Stack Analyzer Console Problems Executa

Monitors + × 0x200011C8 : 0x200011C8 <Hex> × New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
200011C0	00000000	08020080	A5A5A5A5	A5A5A5A5
200011D0	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
200011E0	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
200011F0	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001200	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001210	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001220	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001230	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001240	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001250	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
...				
20001310	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001320	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001330	A5A5A5A5	68130020	A5A5A5A5	A5A5A5A5
20001340	A5A5A5A5	A5A5A5A5	00000000	D4130020
20001350	00000010	04ED00E0	A5A5A5A5	FD2B0008
20001360	DC290008	00000061	A5A5A5A5	F4010000
20001370	A5A5A5A5	00000000	80130020	911A0008
20001380	00080140	20000000	20002000	00080140
20001390	A5A5A5A5	20A00000	F00A0008	A0130020
200013A0	00000020	00000020	A5A5A5A5	00080140
200013B0	20000000	F4010000	A5A5A5A5	F53A0008
200013C0	A5A5A5A5	A5A5A5A5	00000000	C8000080

Explorando el stack

- Y un detalle con las variables locales:

var. conf de TareaParpadeo

Expression	Type	Value
&conf	struct configPa...	0x200013ac <u...
conf	struct configPa...	{...}
puerto	GPIO_TypeDef *	0x40010800
pin	uint32_t	32
periodo	uint32_t	500
Add new e		

Memory × FreeRTOS Task List Static Stack Analyzer Console Problems Executa

Monitors + × 0x200011C8

0x200011C8 : 0x200011C8 <Hex> × New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
200011C0	00000000	08020080	A5A5A5A5	A5A5A5A5
200011D0	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
200011E0	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
200011F0	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001200	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001210	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001220	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001230	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001240	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001250	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
...				
20001310	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001320	A5A5A5A5	A5A5A5A5	A5A5A5A5	A5A5A5A5
20001330	A5A5A5A5	68130020	A5A5A5A5	A5A5A5A5
20001340	A5A5A5A5	A5A5A5A5	00000000	D4130020
20001350	00000010	04ED00E0	A5A5A5A5	FD2B0008
20001360	DC290008	00000061	A5A5A5A5	F4010000
20001370	A5A5A5A5	00000000	80130020	911A0008
20001380	00080140	20000000	20002000	00080140
20001390	A5A5A5A5	20A00000	F00A0008	A0130020
200013A0	00000020	00000020	A5A5A5A5	00080140
200013B0	20000000	F4010000	A5A5A5A5	F53A0008
200013C0	A5A5A5A5	A5A5A5A5	00000000	C8000080

000001F4
00000020

Explorando el stack

- Y el TCB ?

watch expression `pxCurrentTCB`

✓ ➤ <code>pxCurrentTCB</code>	<code>TCB_t * volatile</code>	<code>0x200013d0 <ucH</code>
> ➤ <code>pxTopOfStack</code>	<code>volatile StackType_t *</code>	<code>0x20001328 <ucH</code>
> <code>xStateListItem</code>	<code>ListItem_t</code>	<code>{...}</code>
> <code>xEventListItem</code>	<code>ListItem_t</code>	<code>{...}</code>
(x)= <code>uxPriority</code>	<code>UBaseType_t</code>	<code>8</code>
> ➤ <code>pxStack</code>	<code>StackType_t *</code>	<code>0x200011c8 <ucH</code>
> <code>pcTaskName</code>	<code>char [16]</code>	<code>0x20001404 <ucH</code>
(x)= <code>uxTCBNumber</code>	<code>UBaseType_t</code>	<code>2</code>
(x)= <code>uxTaskNumber</code>	<code>UBaseType_t</code>	<code>0</code>
(x)= <code>uxBasePriority</code>	<code>UBaseType_t</code>	<code>8</code>
(x)= <code>uxMutexesHeld</code>	<code>UBaseType_t</code>	<code>0</code>

Monitor de memoria

Address	0 - 3	4 - 7	8 - B	C - F
200013C0	????	???	???	???
200013D0	(??)	δ???	h?	h?
200013E0	θ??	`?	0???	0???
200013F0	0???	θ??	0???	0???
20001400	È??	tare	aPar	pade
20001410	o???	???	0???	0???
20001420	0???	0???	0E??	PE??
20001430	0E??	0???	0???	0???
20001440	0???	0???	0???	0???
20001450	0???	0???	0???	0???
20001460	0???	0???	0???	0???
20001470	0???	0???	0???	0???
20001480	0???	0???	0???	0???
20001490	0???	0 ??	????	????

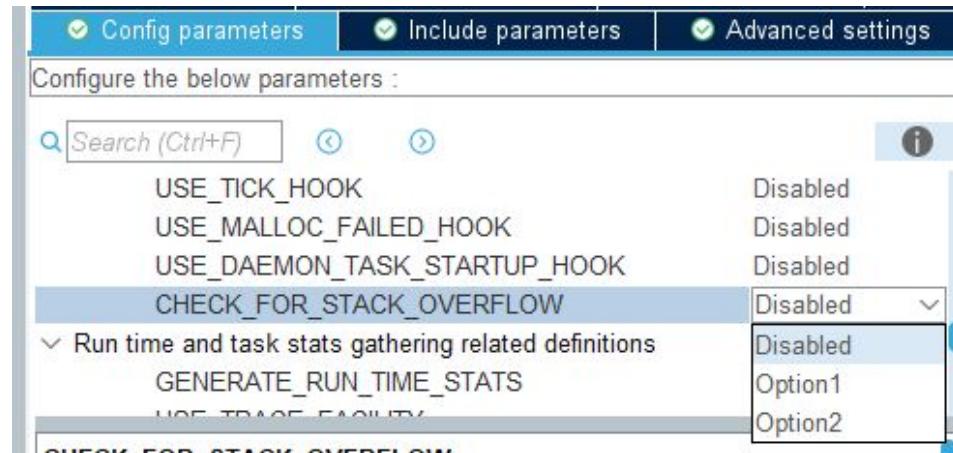
TBC

Comienzo de stack de la tarea

Más herramientas para detectar errores de stack

- Se puede configurar la detección de desborde de stack “automática”
- Opción 1: Chequea si el puntero de stack excede el frame asignado al momento del cambio de contexto
- Opción 2: Hace lo mismo que la opción 1, pero además chequea si los últimos bytes del stack siguen siendo “A5” o se sobrescribieron
- En caso de detectar desborde, se llama un callback:

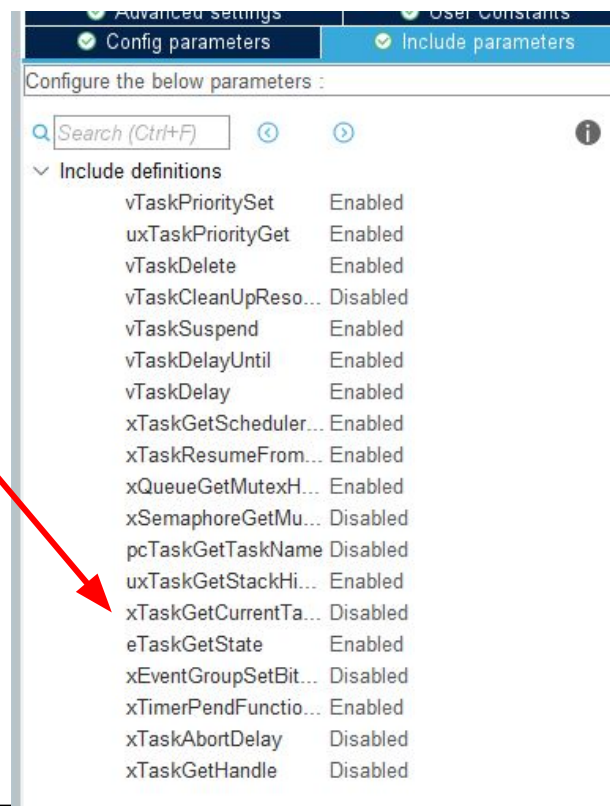
```
void vApplicationStackOverflowHook(  
xTaskHandle xTask, signed char *pcTaskName );
```



Más herramientas para detectar errores de stack

- En tiempo de ejecución el sistema puede obtener información de la cantidad de palabras libres en el stack invocando a la función **UBaseType_t uxTaskGetStackHighWaterMark(TaskHandle_t xTask);**
- INCLUDE_uxTaskGetStackHighWaterMark** tiene que estar seteado en 1 para que esta función esté disponible
- En CMSIS V2 se usa a través de la función :

```
uint32_t osThreadGetStackSpace (osThreadId_t thread_id) {  
    TaskHandle_t hTask = (TaskHandle_t)thread_id;  
    uint32_t sz;  
  
    if (IS_IRQ() || (hTask == NULL)) {  
        sz = 0U;  
    } else {  
        sz = (uint32_t)uxTaskGetStackHighWaterMark (hTask);  
    }  
}
```



Vemos un ejemplo:

Tasks			
Task Name	Pr...	Stack Siz...	Entry Function
defaultTask	o...	128	StartDefaultTask
myTaskLectura	o...	128	entryTaskLectura
myTaskControlad	o...	128	entryTaskControlada

```
void entryTaskLectura(void *argument) {
    /* USER CODE BEGIN entryTaskLectura */
    /* Infinite loop */
    for (;;) {
        libres_lectura =
            4*osThreadGetStackSize(myTaskLecturaHandle);
        libres_controlada =
            4*osThreadGetStackSize(myTaskControladHandle);
        libres_heap = xPortGetFreeHeapSize();
        osDelay(1000);
    }
    /* USER CODE END entryTaskLectura */
}
```

```
/* USER CODE BEGIN PV */
uint32_t libres_lectura;
uint32_t libres_controlada;
uint32_t libres_heap;
/* USER CODE END PV */
```

```
void entryTaskControlada(void *argument) {
    /* USER CODE BEGIN entryTaskControlada */
    /* Infinite loop */
    int n = 0;
    for (;;) {
        n += 10;
        func(n);
        osDelay(2000);
    }
    /* USER CODE END entryTaskControlada */
}
```


Vemos un ejemplo:

```
void entryTaskControlada(void *argument) {
    /* USER CODE BEGIN entryTaskControlada */
    /* Infinite loop */
    int n = 0;
    for (;;) {
        n += 10;
        func(n);
        osDelay(2000);
    }
    /* USER CODE END entryTaskControlada */
}

/* USER CODE BEGIN 4 */
void func(int n) {
    int arr[n];

    int i;
    for (i = 0; i < n; i++) {
        arr[i] = 10;
    }
    return;
}
```

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlada	uint32_t	352
(x)= libres_heap	uint32_t	1192

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlada	uint32_t	152
(x)= libres_heap	uint32_t	1192

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlada	uint32_t	32
(x)= libres_heap	uint32_t	1192

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlada	uint32_t	0
(x)= libres_heap	uint32_t	1192

```
83  * @brief This function handles HardFault_IRQn interrupt
84  */
85  void HardFault_Handler(void)
86  {
87      /* USER CODE BEGIN HardFault_IRQn 0 */
88
89      /* USER CODE END HardFault_IRQn 0 */
90      while (1)
91      {
92          /* USER CODE BEGIN W1_HardFault_IRQn 0 */
93          /* USER CODE END W1_HardFault_IRQn 0 */
94      }

```

Usando malloc

```
void entryTaskControlada(void *argument) {
    /* USER CODE BEGIN entryTaskControlada */
    /* Infinite loop */
    int n = 0;
    for (;;) {
        n += 10;
        func(n);
        osDelay(2000);
    }
    /* USER CODE END entryTaskControlada */
}
```

```
void func(int n) {

    int *arr = (int*) pvPortMalloc(sizeof(int)*n);
    int i;
    for (i = 0; i < n; i++) {
        arr[i] = 10;
    }
    return;
}
```

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlad	uint32_t	372
(x)= libres_heap	uint32_t	1144

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlad	uint32_t	372
(x)= libres_heap	uint32_t	928

Expression	Type	Value
(x)= libres_lectura	uint32_t	360
(x)= libres_controlad	uint32_t	372
(x)= libres_heap	uint32_t	0

```
85 void HardFault_Handler(void)
86 {
87     /* USER CODE BEGIN HardFault_IRQn 0 */
88
89     /* USER CODE END HardFault_IRQn 0 */
90     while (1)
91     {
```

Malloc permite detectar fallos

```
void entryTaskControlada(void *argument) {  
    /* USER CODE BEGIN entryTaskControlada */  
    /* Infinite loop */  
    int n = 0;  
    for (;;) {  
        n += 10;  
        func(n);  
        osDelay(2000);  
    }  
    /* USER CODE END entryTaskControlada */  
}
```

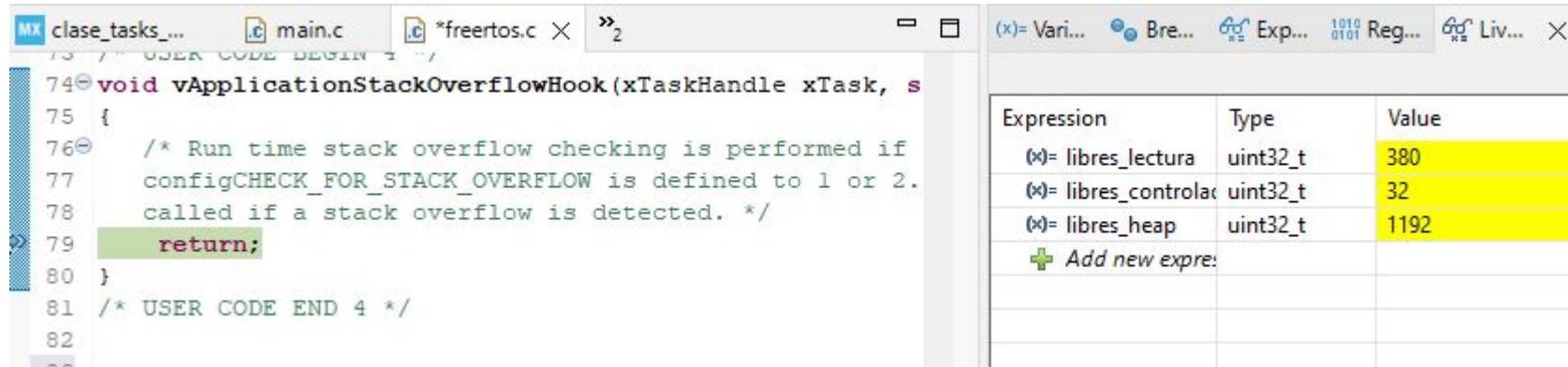
```
228 /* USER CODE BEGIN 4 */  
229 void func(int n) {  
230  
231     int *arr = (int*) pvPortMalloc(sizeof(int)*n);  
232     int i;  
233  
234     if(arr == NULL ) {  
235         return;  
236     }
```

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlad	uint32_t	372
(x)= libres_heap	uint32_t	0
+ Add new expre:		

Volviendo al ejemplo de stack overflow

```
void entryTaskControlada(void *argument) {  
    /* USER CODE BEGIN entryTaskControlada */  
    /* Infinite loop */  
    int n = 0;  
    for (;;) {  
        n += 10;  
        func(n);  
        osDelay(2000);  
    }  
    /* USER CODE END entryTaskControlada */  
}
```

```
/* USER CODE BEGIN 4 */  
void func(int n) {  
    int arr[n];  
  
    int i;  
    for (i = 0; i < n; i++) {  
        arr[i] = 10;  
    }  
    return;  
}
```



The screenshot shows an IDE with two code files open: `main.c` and `*freertos.c`. The `*freertos.c` file is active, showing the `vApplicationStackOverflowHook` function. The function is currently at line 79, where `return;` is highlighted. The function body includes a comment about stack overflow checking and a `return;` statement.

On the right side, there is a variable watch window titled `(x)= Vari...`. It contains a table with the following data:

Expression	Type	Value
(x)= libres_lectura	uint32_t	380
(x)= libres_controlad	uint32_t	32
(x)= libres_heap	uint32_t	1192
+ Add new expres...		

Prioridades de las tareas

Prioridades

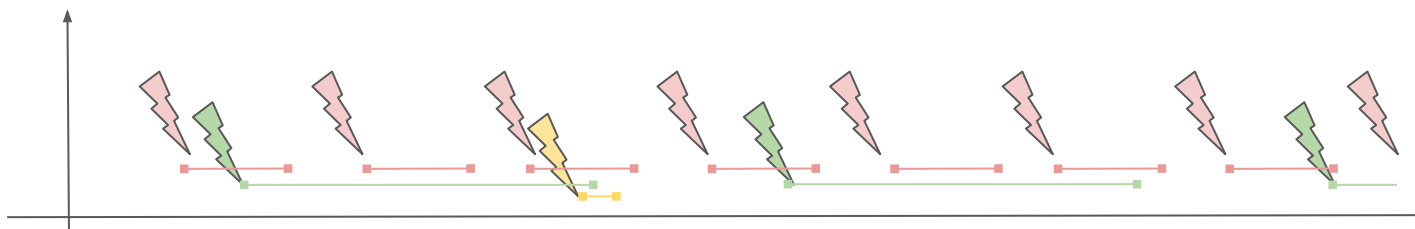
- CMSIS-RTOS v2, tiene un esquema de prioridades con 8 niveles, cada uno a su vez subdividido en 8
- Esos niveles se mapean a prioridades de FreeRTOS según se encuentra en cmsis_os2.h
- `osThreadSetPriority()` permite cambiar la prioridad de un thread y `osThreadGetPriority()` obtener su valor
- La función que ya comentamos, `osThreadYield()`, sólo cederá la ejecución a una tarea de mayor prioridad, pero no a una de menor prioridad aunque la misma esté READY

```
/// Priority values.
typedef enum {
    osPriorityNone           = 0,
    osPriorityIdle           = 1,
    osPriorityLow            = 8,
    osPriorityLow1          = 8+1,
    ...
    osPriorityLow7           = 8+7,
    osPriorityBelowNormal    = 16,
    ...
    osPriorityBelowNormal7  = 16+7,
    osPriorityNormal        = 24,
    ...
    osPriorityNormal7       = 24+7,
    osPriorityAboveNormal    = 32,
    ...
    osPriorityAboveNormal7  = 32+7,
    osPriorityHigh          = 40,
    ...
    osPriorityHigh7         = 40+7,
    osPriorityRealtime       = 48,
    ...
    osPriorityRealtime7     = 48+7,
    osPriorityISR            = 56,
    osPriorityError         = -1,
} osPriority_t;
```

Eventos e Interrupciones

Atención de eventos

- Al trabajar con sistemas embebidos de tiempo real lidiamos con la respuesta en tiempos acotados a eventos (vimos latencia y jitter)
- Estos eventos pueden ser comunicados a través de periféricos o disparados por software
- La forma de notificar al sistema de su ocurrencia puede ser por interrupciones o por polling
- Podemos usar tareas e interrupciones como herramienta para programar la respuesta a eventos. ¿Cuánto hay que hacer en la tarea, y cuanto en el cuerpo de la interrupción? ¿Cómo se pueden comunicar entre sí? ¿Cómo organizar la ejecución?



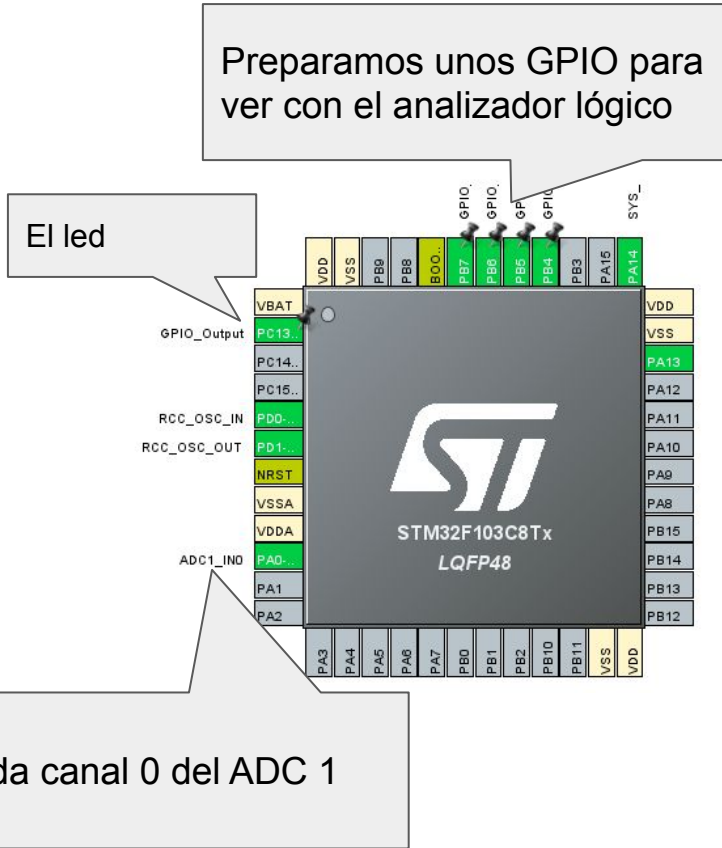
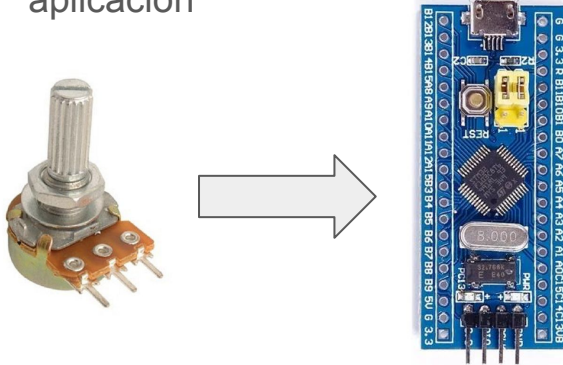
Interrupciones y prioridades

- Recordemos que las tareas tienen prioridades, y las interrupciones también tienen prioridades
- Las prioridades de las tareas le dicen al scheduler cuál poner a correr a continuación, dentro de un esquema de software que no está relacionada con el hardware
- Las interrupciones tienen una prioridad con efecto o resolución a nivel del Hardware (Procesador, NVIC y otros periféricos)
- Sin embargo, **ambos “mundos” se relacionan ya que el sistema operativo realiza el cambio de contexto a partir del uso de interrupciones.**



Volvemos a prioridades: ejemplo

- Se tiene una señal de entrada que simularemos con un potenciómetro
- Se quiere adquirir a una tasa de 20kHz (50 μ s de período) para obtener una tasa final de 2 kHz realizando un promedio de 10 muestras
- Se usa FreeRTOS en paralelo a esta aplicación



Preparamos la aplicación

Instalación de Tareas

Tasks and Queues Timers and Semaphores

Config parameters Include parameters

Tasks

Task Name	Priority	Sta...	Entry Function
TaskNormal	osPriorityNormal	128	entryTaskNormal
TaskAlta	osPriorityHigh	128	entryTaskAlta

Configuración de interrupción timer 2

RTC
TIM1
TIM2
TIM3
TIM4

Connectivity >

Parameter Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits value) 71

Counter Mode Up

Counter Period (AutoReload ... 49

Internal Clock Division (CKD) No Division

auto-reload preload Enable

Timers

RTC
TIM1
TIM2
TIM3
TIM4

User Constants NVIC S

Parameter

NVIC Interrupt Table Enabled

TIM2 global interrupt ☒

Preparamos la aplicación

Debug con TaskHooks

```
/* FreeRTOSConfig.h*/  
void callback_in(int);  
void callback_out(int);  
#define traceTASK_SWITCHED_IN() callback_in((int)pxCurrentTCB->pxTaskTag)  
#define traceTASK_SWITCHED_OUT() callback_out((int)pxCurrentTCB->pxTaskTag)  
#define TAG_TASK_IDLE 0  
#define TAG_TASK_NORMAL 1  
#define TAG_TASK_ALTA 2
```

```
/* En cada tarea (incluir task.h)*/  
vTaskSetApplicationTaskTag(NULL, (void*) TAG_TASK_NORMAL);
```

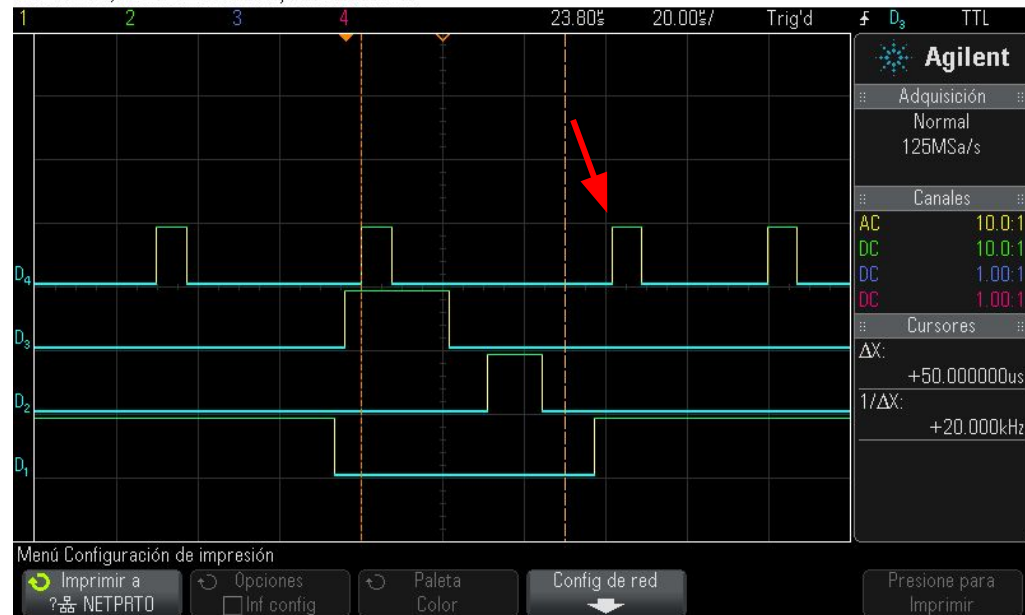
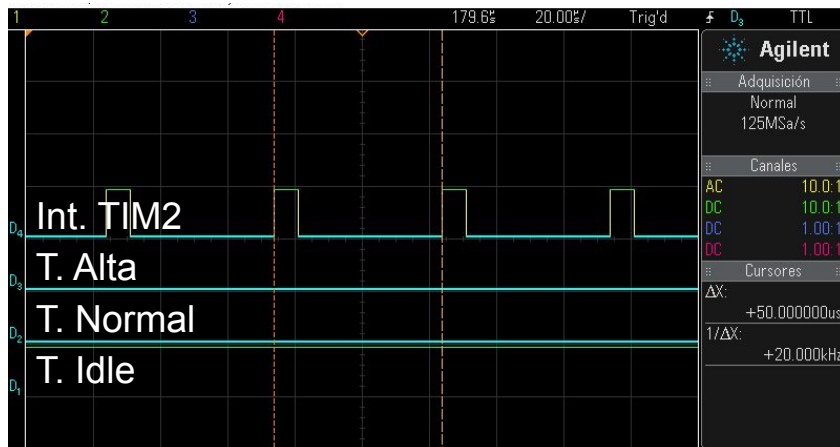
```
void callback_in(int tag) { /* Definición en main.c*/  
    switch (tag) {  
        case TAG_TASK_IDLE: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);break;  
        case TAG_TASK_NORMAL: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);break;  
        case TAG_TASK_ALTA: HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);break;  
    }  
}
```

Preparamos la aplicación

En la interrupción del timer disparamos la conversión y tomamos la muestra

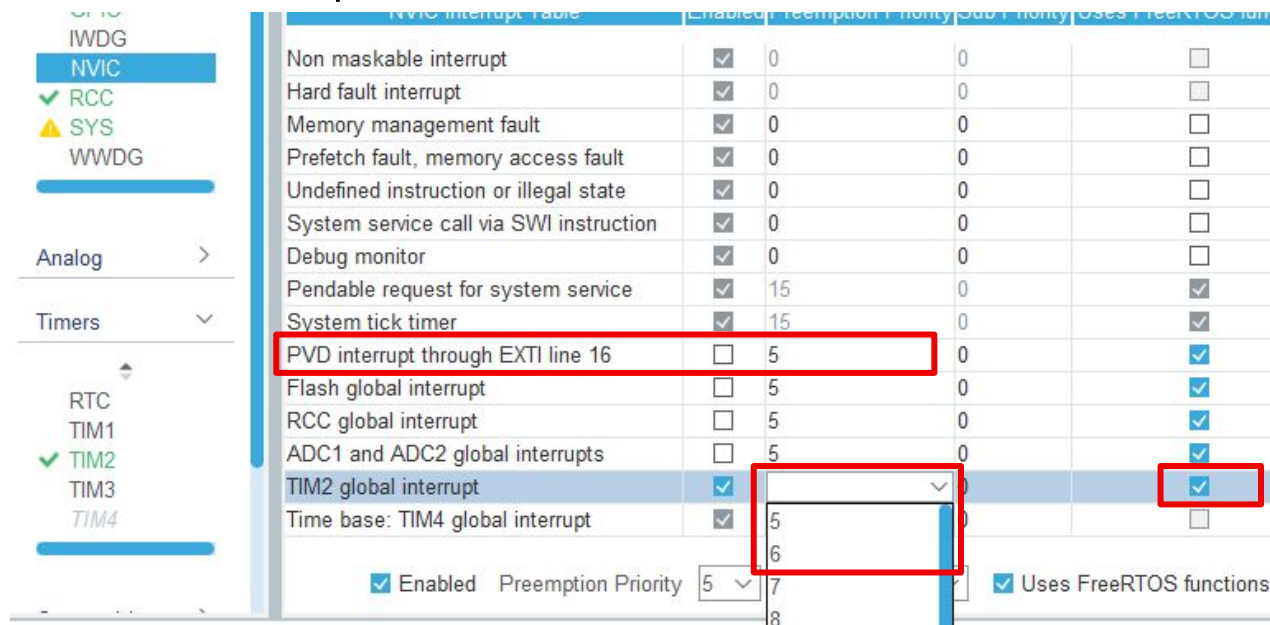
```
void HAL_TIM_PeriodElapsedCallbackTIM2 (TIM_HandleTypeDef *htim) {  
    if (htim->Instance == TIM2) {  
        HAL_GPIO_WritePin (GPIOB, GPIO_PIN_7, GPIO_PIN_SET);  
        HAL_ADC_Start (&hadc1);  
        HAL_ADC_PollForConversion (&hadc1, 100);  
        val = HAL_ADC_GetValue (&hadc1);  
        HAL_GPIO_WritePin (GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);  
    }  
}
```

Veamos la salida:



En este caso, el esquema de prioridades ocasionó un incumplimiento del período.
Vamos a analizar por qué

Niveles de prioridad



Interrupt	Enabled	Preemption Priority	Sub Priority	Uses FreeRTOS functions
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Memory management fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Debug monitor	<input checked="" type="checkbox"/>	0	0	<input type="checkbox"/>
Pendable request for system service	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
System tick timer	<input checked="" type="checkbox"/>	15	0	<input checked="" type="checkbox"/>
PVD interrupt through EXTI line 16	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
Flash global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
RCC global interrupt	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
ADC1 and ADC2 global interrupts	<input type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
TIM2 global interrupt	<input checked="" type="checkbox"/>	5	0	<input checked="" type="checkbox"/>
Time base: TIM4 global interrupt	<input checked="" type="checkbox"/>	5	0	<input type="checkbox"/>

(3) Esos valores son configurables, y la limitación sobre el nivel de la interrupción se debe a una configuración activada por default

(1) La prioridad de interrupción del timer está limitada en 5 en el entorno gráfico

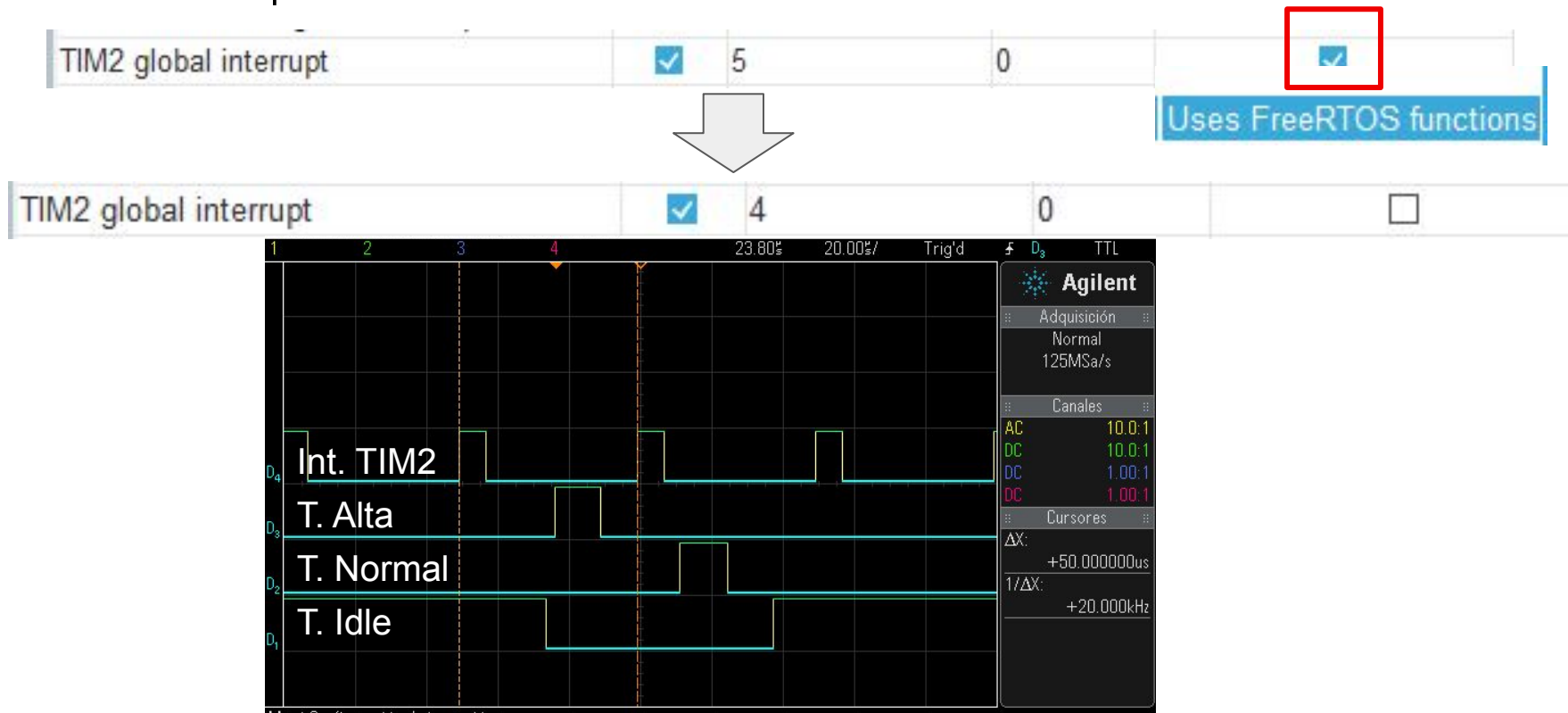
(2) Es el mismo nivel de prioridad del PVD que se llama para producir un cambio de contexto

Niveles de prioridad

```
▼ Interrupt nesting behaviour configuration
    LIBRARY_LOWEST_INTERRUPT_PRIORITY    15
    LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5
```

- **LIBRARY_LOWEST_INTERRUPT_PRIORITY**
La prioridad más baja asignable (siguiendo en este caso la sintaxis del Cortex M3 con 16 niveles)
- **LIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY**
La mayor prioridad que se puede asignar que va a seguir estando dentro del esquema de funcionamiento del sistema operativo, y por lo tanto va a permitir llamar funciones de la API compatibles con interrupciones. Si se asigna una prioridad mayor (número más bajo) ya no podrán usarse funciones de la API

Niveles de prioridad



Niveles de prioridad

- Resolvimos la especificación de tiempo real al disparar la conversión en el tiempo adecuado
- Pero ahora queremos enviar el dato capturado a una tarea para ser procesado, y nos encontramos que estamos fuera del RTOS
- Podemos distribuir de otra manera el resto del procesamiento.
- Por ejemplo, podemos usar la interrupción del ADC, pero sin aumentarle tanto la prioridad



ADC1 and ADC2 global interrupts	<input checked="" type="checkbox"/>	5
TIM2 global interrupt	<input checked="" type="checkbox"/>	4

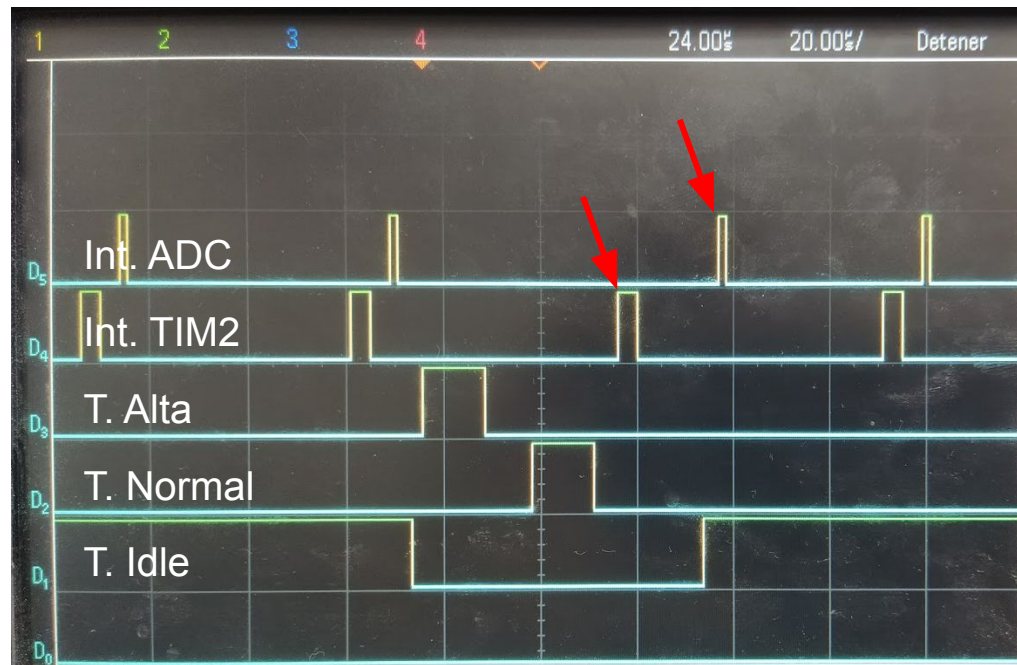
Niveles de prioridad

```
void HAL_TIM_PeriodElapsedCallbackTIM2(TIM_HandleTypeDef *htim) {  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_SET);  
    HAL_ADC_Start_IT(&hadc1);  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);  
}
```

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);  
    int16_t val;  
    val = HAL_ADC_GetValue(&hadc1);  
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);  
}
```

ADC1 and ADC2 global interrupts	<input checked="" type="checkbox"/>	5
TIM2 global interrupt	<input checked="" type="checkbox"/>	4

Niveles de prioridad



ADC1 and ADC2 global interrupts



5

TIM2 global interrupt

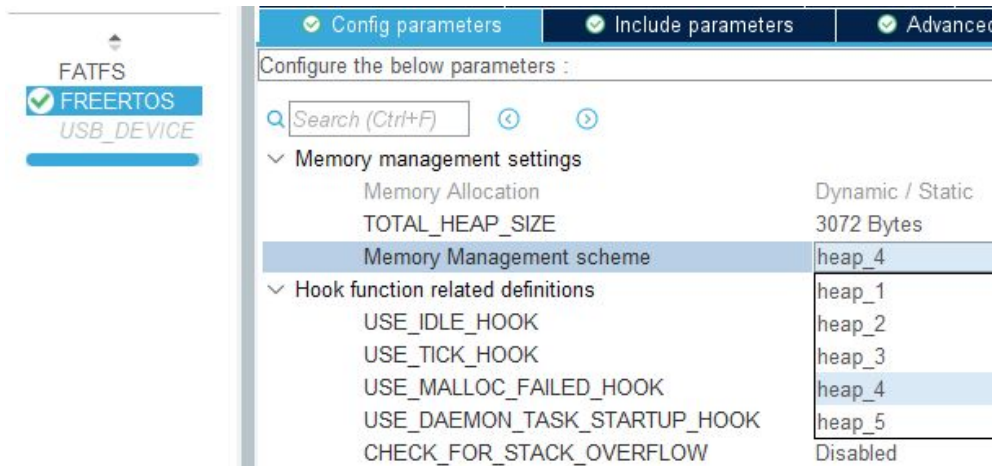


4

Manejo de Memoria

Asignación de memoria

- Las tareas pueden crearse y borrarse dinámicamente, según el esquema de asignación de memoria elegido
- ¿Quién se encarga de asignar memoria?
- ¿Y si, como suele suceder, necesitamos asignar memoria dinámica para nuestras variables?
- Existen distintos esquemas que pueden elegirse



Asignación de memoria

- **heap_1.c**

- Es el esquema más simple y no permite que se libere memoria una vez que se asignó.
- Subdivide un único arreglo en bloques cuando se realizan pedidos de espacio en RAM
- El tamaño total del arreglo se define a partir de configTOTAL_HEAP_SIZE en FreeRTOSConfig.h.
- Se puede utilizar si no se eliminan elementos (no hay llamadas a vTaskDelete () o vQueueDelete () por ejemplo)
- Siempre es determinista: toma siempre el mismo tiempo para reservar un bloque.

- **heap_2.c**

- Este esquema busca entre la memoria libre y elige el bloque que mejor se ajusta al tamaño necesario (best-fit).
- Además, puede liberar bloques (permite usar vTaskDelete() y vQueueDelete())
- Sin embargo, no puede juntar bloques que se habían reservado previamente aunque se liberen, lo cual puede conducir a la fragmentación de la memoria.
- El tamaño total de RAM disponible se configura con configTOTAL_HEAP_SIZE
- No es determinista ya que depende del algoritmo que encuentra el mejor bloque disponible, pero al ser un algoritmo sencillo pueden predecirse tiempos máximos con cotas optimistas.

Asignación de memoria

- **heap_4.c**

- El algoritmo asigna el primer bloque suficientemente grande para la memoria necesaria, combinando bloques libres adyacentes
- Es conveniente cuando se repite la creación y borrado de tareas y queues, pero también cuando se requieren numerosos llamados a vPortMalloc() and vPortFree()
- No es determinista, pero vPortMalloc/Free son más eficientes que malloc/free estándar de las librerías de C
- Por este conjunto de características, es el algoritmo seleccionado por defecto

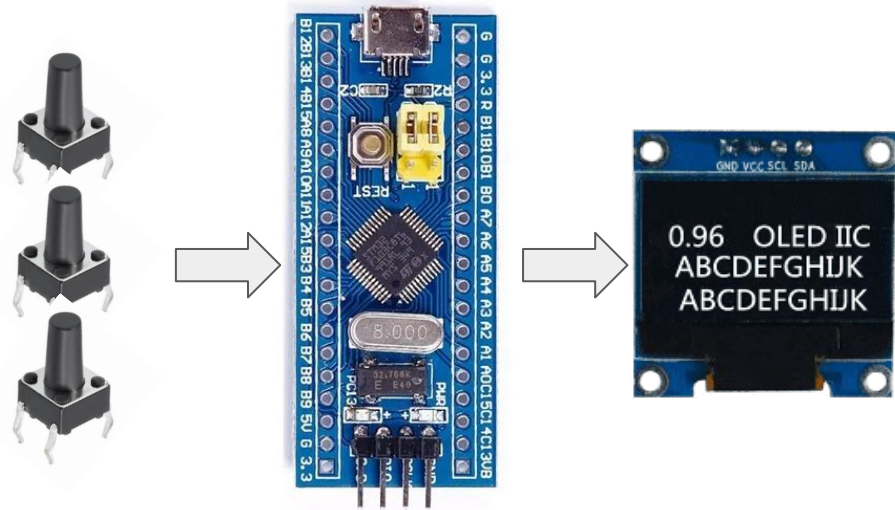
- **Asignación estática**

- En lugar de algoritmos dinámicos puede seleccionarse la opción estática y se asignan bloques de un arreglo programando las direcciones.

Consigna-Ejemplo

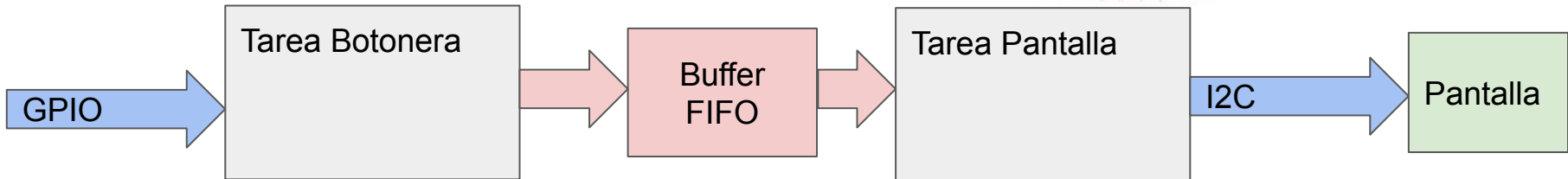
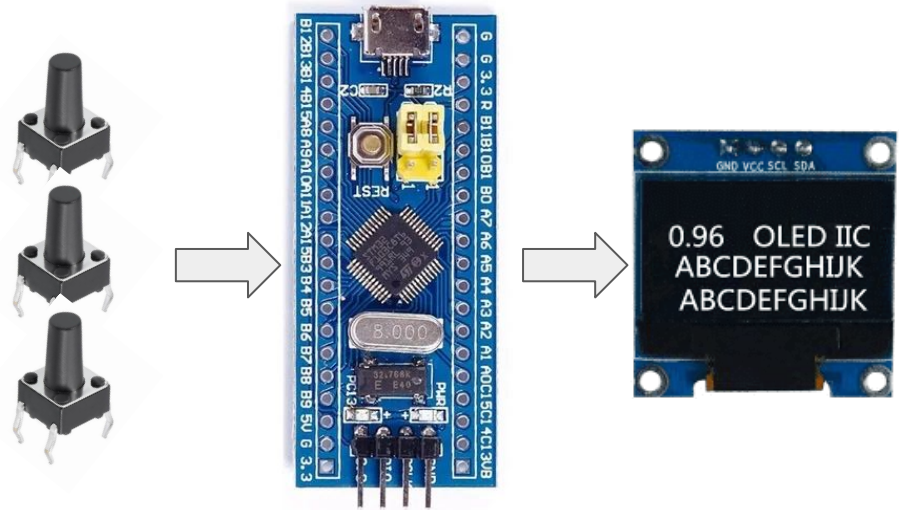
Un programa de ejemplo

- Se presionan botones y se imprimen en la pantalla consecutivamente los botones presionados
- Se hará a partir de 2 tareas:
 - Una de control de la botonera
 - Una de control de la pantalla
- La tarea de la botonera deberá registrar los botones presionados y enviar la información a la tarea de impresión
- Cumpliremos algunos requisitos

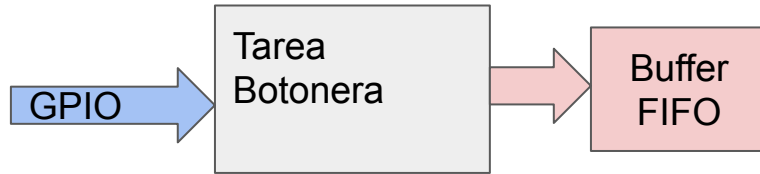


Un programa de ejemplo

- Cada botón se considerará activado luego de haber soltado el botón (no mientras se lo presiona)
- El dato del botón activado se almacenará con un enum que contendrá un elemento para cada botón que exista: izquierda, centro, derecha, ninguno
- La comunicación se dará a través de un buffer FIFO



Un programa de ejemplo



```
enum botones_e { UP, IZQ, DER, NINGUNO};
```

```
enum botones_e leerBoton(void)
```

```
void entryTareaBotonera(void *argument) {
```

```
    enum botones_e boton;
```

```
    enum { btn_soltando, btn_reposo
```

```
    } estado_btn = btn_reposo;
```

```
    for (;;) {
```

```
        switch (estado_btn) {
```

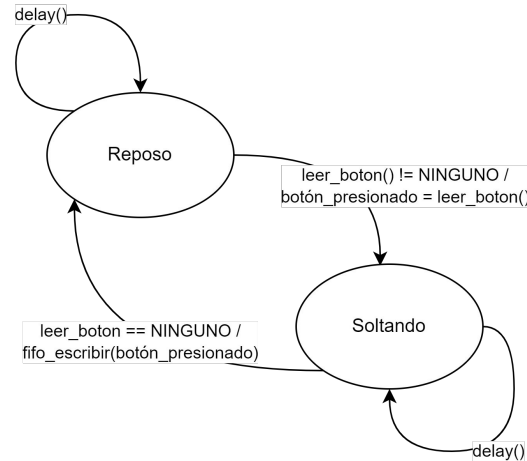
```
            ...
```

```
        }
```

```
        osDelay( ... );
```

```
    }
```

```
}
```

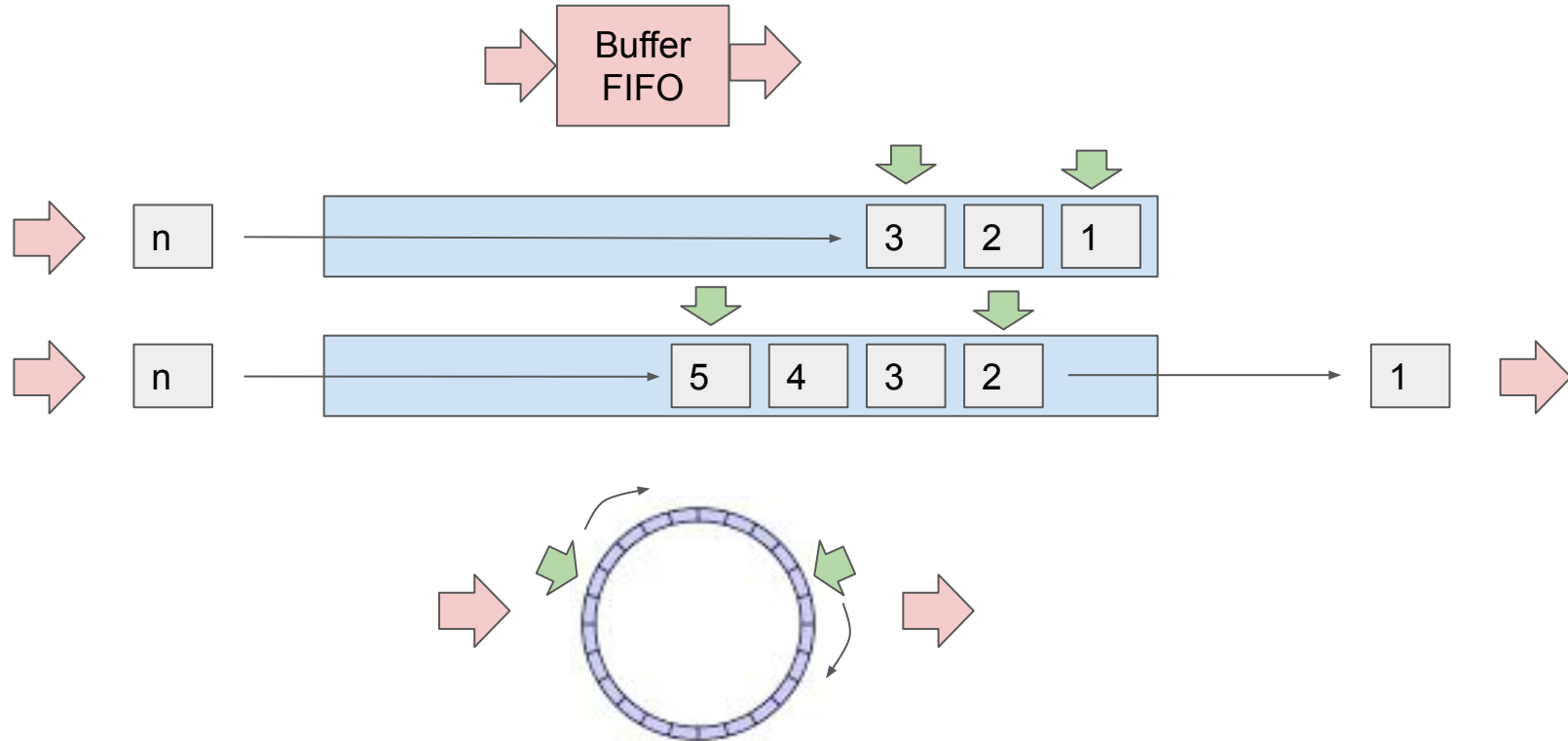


Un programa de ejemplo

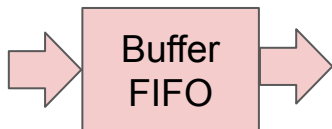


```
void entryTareaPantalla(void *argument) {  
    SSD1306_Init ();  
    SSD1306_Clear();  
    SSD1306_UpdateScreen();  
    for (;;) {  
        if(mififo_haydatos()){  
            // Escribir pantalla según dato  
        }  
        osDelay( ... );  
    }  
}
```

Un programa de ejemplo



Un programa de ejemplo



```
#define MF_TAM 20
typedef char mififo_tipo_t;

struct mififo_t {
    mififo_tipo_t arr[MF_TAM];
    int idxr;
    int idxw;
};

int mififo_haydatos(struct mififo_t *mf) {
    return mf->idxr != mf->idxw;
}
```

```
void mififo_cargardato(struct mififo_t *mf, mififo_tipo_t dato) {
    int prox = (mf->idxw+1)%MF_TAM;
    if(mf->idxr == prox) {
        return;
    }
    else {
        mf->arr[mf->idxw] = dato;
        mf->idxw = prox;
    }
}

mififo_tipo_t mififo_leerdato(struct mififo_t *mf) {
    if(mf->idxw == mf->idxr) {
        return -1;
    }
    mififo_tipo_t dato = mf->arr[mf->idxr];
    mf->idxr = (mf->idxr+1)%MF_TAM;
    return dato;
}
```