

# Mbed OS

## The Things Network Madrid

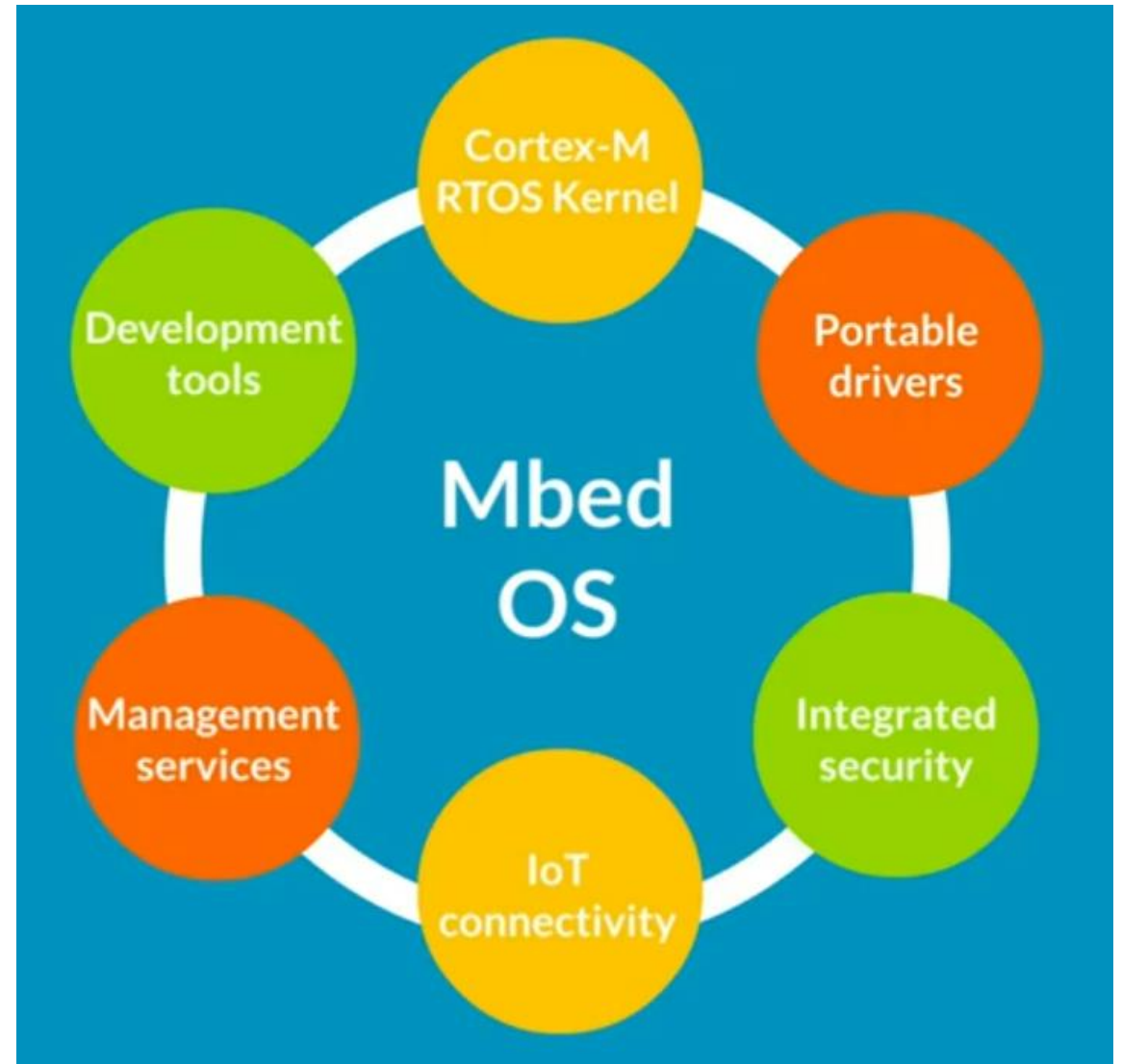
---

Juan Félix Mateos  
Octubre 2021  
[juanfelixmateos@gmail.com](mailto:juanfelixmateos@gmail.com)

[juanfelixmateos@gmail.com](mailto:juanfelixmateos@gmail.com)

# ¿Qué es un RTOS?

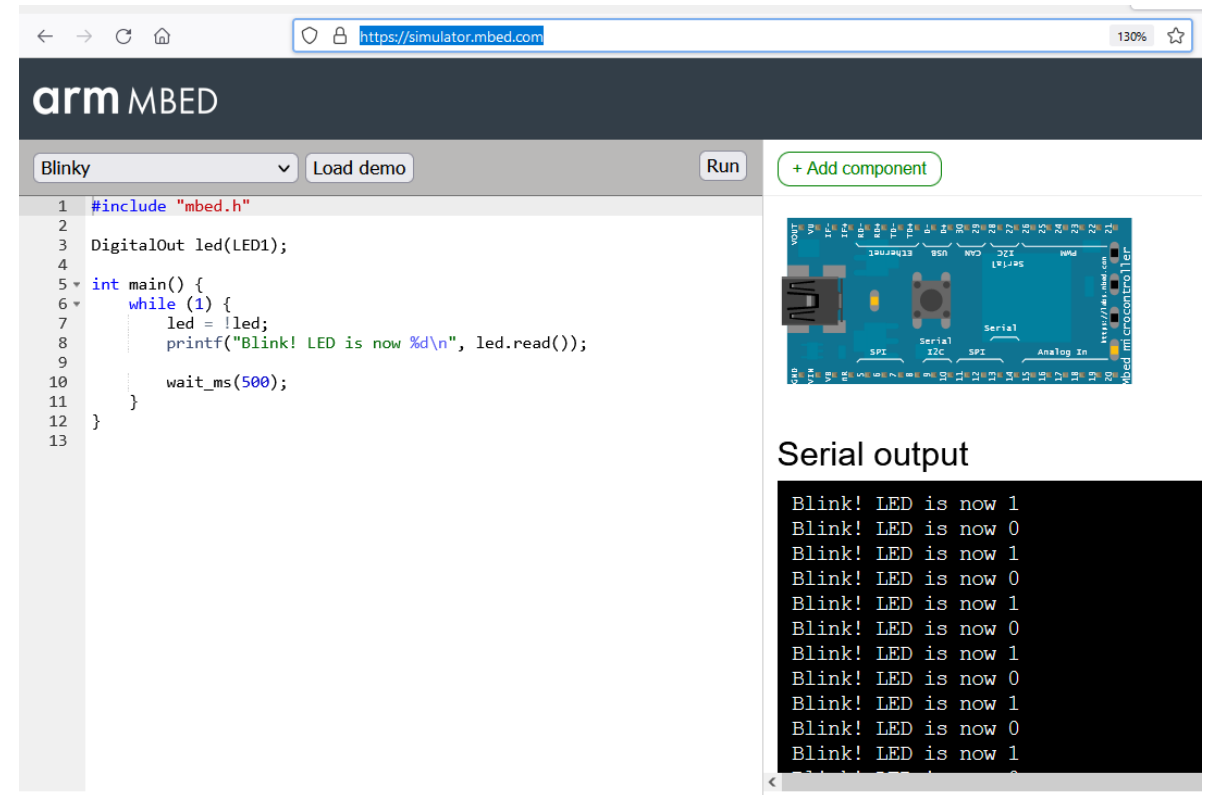
- Sistema operativo de tiempo real
- Capa de abstracción entre el hardware y el programador
- **Determinista:** Cada operación tiene un tiempo fijo asignado para ejecutarse (o fallar).
- Conceptos:
  - Semáforos
  - Locks/Mutexes
  - Multi-Treading



# Mbed simulator

<https://simulator.mbed.com/>  
Experimental  
Mbed OS 5  
No admite threads

Si falla, insistir pulsando  
nuevamente el botón download  
que hay a la derecha de Add  
component



# GPIO

Table 19. STM32WLE5/E4xx pin definition (continued)

Pin number			Pin name (function after reset)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
UFQFPN48	WLCSP59	UFBGA73						
11	K11	H5	VDD	S	-	-	-	-
12	J10	J1	PA4	I/O	FT	-	RTC_OUT2, LPTIM1_OUT, SPI1_NSS, USART2_CK, DEBUG_SUBGHZSPI_ NSSOUT, LPTIM2_OUT, CM4_EVENTOUT	-
13	H9	J2	PA5	I/O	FT	-	TIM2_CH1, TIM2_ETR, SPI2_MISO, SPI1_SCK, DEBUG_SUBGHZSPI_ SCKOUT, LPTIM2_ETR, CM4_EVENTOUT	-
14	G8	F4	PA6	I/O	FT	-	TIM1_BKIN, I2C2_SMBA, SPI1_MISO, LPUART1_CTS, DEBUG_SUBGHZSPI_ MISOOUT, TIM16_CH1, CM4_EVENTOUT	-
15	E8	H3	PA7	I/O	FT_fa	-	TIM1_CH1N, I2C3_SCL, SPI1_MOSI, COMP2_OUT, DEBUG_SUBGHZSPI_ MOSIOUT, TIM17_CH1, CM4_EVENTOUT	-

## GPIO

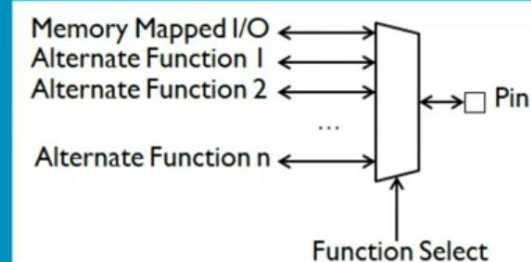
General Purpose **Input** **Output**

Configurable for a range of signals

### Advantages

Saves space

Improves flexibility



# API: Clases y métodos

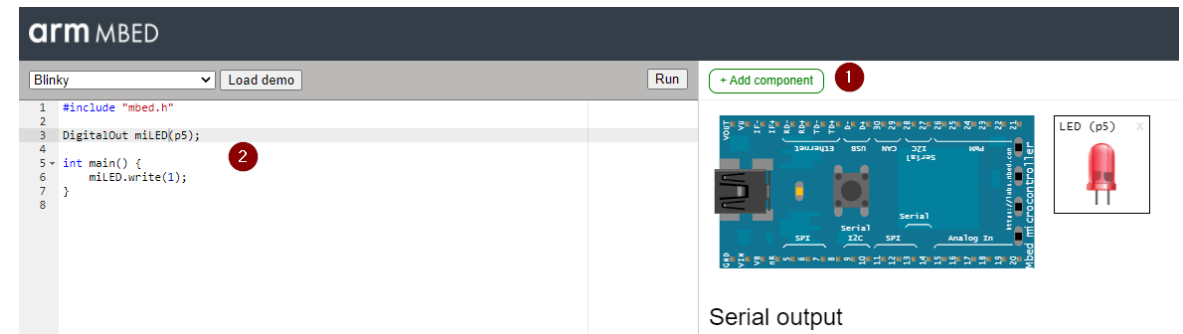
<https://os.mbed.com/docs/mbed-os/v6.15/apis/digitalout.html>

## DigitalOut class reference

### DigitalOut Class Reference

#### Public Member Functions

	<a href="#">DigitalOut (PinName pin)</a>
	Create a <a href="#">DigitalOut</a> connected to the specified pin. <a href="#">More...</a>
	<a href="#">DigitalOut (PinName pin, int value)</a>
	Create a <a href="#">DigitalOut</a> connected to the specified pin. <a href="#">More...</a>
void	<a href="#">write (int value)</a>
	Set the output, specified as 0 or 1 (int) <a href="#">More...</a>



The screenshot shows the mbed IDE interface. On the left, the code editor displays the following code:

```
1 #include "mbed.h"
2
3 DigitalOut mLED(p5);
4
5 int main() {
6     mLED.write(1);
7 }
8
```

Red circles with numbers 1 and 2 highlight the components: circle 1 points to the '+ Add component' button, and circle 2 points to the `mLED.write(1);` line in the code.

On the right, the hardware configuration panel shows a blue mbed microcontroller board with various components connected. A red LED is connected to pin p5, labeled 'LED (p5)'. Below the hardware panel, the 'Serial output' section is visible.

# Nomenclatura de los pines

- El emulador está basado en el NXP LPC1768
- [https://github.com/ARMmbed/mbed-os/blob/master/targets/TARGET\\_NXP/TARGET\\_LPC176X/TARGET\\_MBED\\_LPC1768/PinNames.h](https://github.com/ARMmbed/mbed-os/blob/master/targets/TARGET_NXP/TARGET_LPC176X/TARGET_MBED_LPC1768/PinNames.h)
- Podríamos cambiar en el código anterior p5 por P0\_9 y funcionaría igual

```
30     PIN_INPUT,  
31     PIN_OUTPUT  
32 } PinDirection;  
33  
34 #define PORT_SHIFT 5  
35  
36 typedef enum {  
37     // LPC Pin Names  
38     P0_0 = LPC_GPIO0_BASE,  
39     P0_1, P0_2, P0_3, P0_4, P0_5, P0_6, P0_7, P0_8, P0_9, P0_10, P0_11, P0_12, P0_13, P0_14, P0_15, P0_16, P0_17, P0_18, P0_19, P0_20,  
40     P1_0, P1_1, P1_2, P1_3, P1_4, P1_5, P1_6, P1_7, P1_8, P1_9, P1_10, P1_11, P1_12, P1_13, P1_14, P1_15, P1_16, P1_17, P1_18, P1_19, P1_20,  
41     P2_0, P2_1, P2_2, P2_3, P2_4, P2_5, P2_6, P2_7, P2_8, P2_9, P2_10, P2_11, P2_12, P2_13, P2_14, P2_15, P2_16, P2_17, P2_18, P2_19, P2_20,  
42     P3_0, P3_1, P3_2, P3_3, P3_4, P3_5, P3_6, P3_7, P3_8, P3_9, P3_10, P3_11, P3_12, P3_13, P3_14, P3_15, P3_16, P3_17, P3_18, P3_19, P3_20,  
43     P4_0, P4_1, P4_2, P4_3, P4_4, P4_5, P4_6, P4_7, P4_8, P4_9, P4_10, P4_11, P4_12, P4_13, P4_14, P4_15, P4_16, P4_17, P4_18, P4_19, P4_20,  
44  
45     // MBED OS Pin Names  
46     p5 = P0_9,  
47     p6 = P0_0,  
48     p7 = P0_7,  
49     p8 = P0_6,  
50     p9 = P0_0,  
51     p10 = P0_1,  
52     p11 = P0_18,  
53     p12 = P0_17
```

# DigitalIn

**DigitalIn** (PinName pin)

Create a **DigitalIn** connected to the specified pin. [More...](#)

**DigitalIn** (PinName pin, PinMode mode)

Create a **DigitalIn** connected to the specified pin. [More...](#)

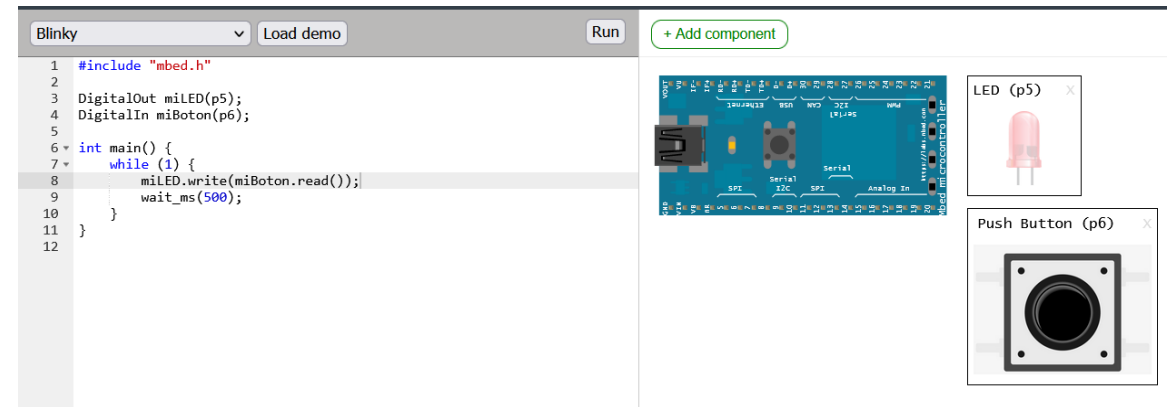
**~DigitalIn** ()

Class destructor, deinitialize the pin. [More...](#)

int **read** ()

Read the input, represented as 0 or 1 (int) [More...](#)

void **mode** (PinMode mode)



Los modos pull son:

- PullUp, PullDown, PullNone, OpenDrain

En el emulador hay que poner siempre wait en los bucles infinitos

# PWMOut

```
1  #include "mbed.h"
2
3  PwmOut miLED(p5);
4
5  int main() {
6      while(1) {
7          for(float i=0;i<1;i=i+0.1){
8              miLED.write(i);
9              wait(0.5);
10         }
11         for(float i=1;i>0;i=i-0.1){
12             miLED.write(i);
13             wait(0.5);
14         }
15     }
16 }
```

## PwmOut Class Reference

### Public Member Functions

	<a href="#">PwmOut</a> (PinName pin)
	Create a <a href="#">PwmOut</a> connected to the specified pin. <a href="#">More...</a>
	<a href="#">PwmOut</a> (const <a href="#">PinMap</a> &pinmap)
	Create a <a href="#">PwmOut</a> connected to the specified pin. <a href="#">More...</a>
void	<a href="#">write</a> (float value)
	Set the output duty-cycle, specified as a percentage (float) <a href="#">More...</a>



# AnalogIn

## FL 14 AnalogIn Class Reference

### Public Member Functions

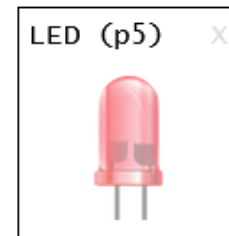
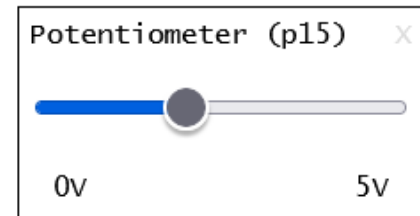
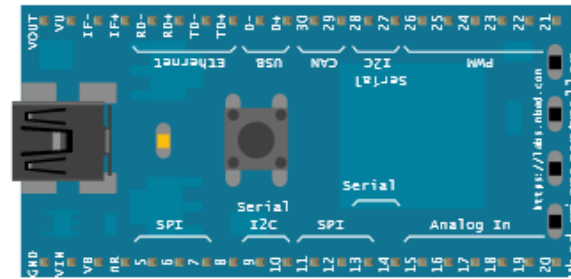
	<a href="#">AnalogIn</a> (const <a href="#">PinMap</a> &pinmap, float vref=MBED_CONF_TARGET_DEFAULT_ADC_VREF)
	Create an <a href="#">AnalogIn</a> , connected to the specified pin. <a href="#">More...</a>
	<a href="#">AnalogIn</a> (PinName pin, float vref=MBED_CONF_TARGET_DEFAULT_ADC_VREF)
	Create an <a href="#">AnalogIn</a> , connected to the specified pin. <a href="#">More...</a>
float	<a href="#">read</a> ()
	Read the input voltage, represented as a float in the range [0.0, 1.0]. <a href="#">More...</a>

Symbol	Pin/ball					
	LQFP100	TFBGA100	WLCSP100			
P0[23]/AD0[0]/ I2SRX_CLK/ CAP3[0]	9	E5	D5	<a href="#">[2]</a>	I/O	<b>P0[23]</b> — General purpose digital input/output pin.
					I	<b>AD0[0]</b> — A/D converter 0, input 0.
					I/O	<b>I2SRX_CLK</b> — Receive Clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the <i>I<sup>2</sup>S-bus</i> specification. (LPC1769/68/67/66/65/63 only).
					I	<b>CAP3[0]</b> — Capture input for Timer 3, channel 0.

No todos los pines tienen funcionalidad ADC.  
El p15 del LPC1768 es el P0\_23, que es la  
entrada 0 del ACD 0.

# AnalogIn y PwmOut

```
1 #include "mbed.h"
2
3 PwmOut miLED(p5);
4 AnalogIn miPot(p15);
5
6 int main() {
7     while (1) {
8         miLED.write(miPot.read());
9         printf("Intensidad: %.2f\n", miLED.read());
10        wait_ms(500);
11    }
12 }
13
```



Serial output

```
Intensidad: 0.39
Intensidad: 0.39
Intensidad: 0.39
```

# Interrupciones externas

## InterruptIn Class Reference

### Public Member Functions

	<a href="#">InterruptIn (PinName pin)</a>
	Create an <a href="#">InterruptIn</a> connected to the specified pin. <a href="#">More...</a>
	<a href="#">InterruptIn (PinName pin, PinMode mode)</a>
	Create an <a href="#">InterruptIn</a> connected to the specified pin, and the pin configured to the specified mode. <a href="#">More...</a>
int	<a href="#">read ()</a>
	Read the input, represented as 0 or 1 (int) <a href="#">More...</a>
	<a href="#">operator int ()</a>
	An operator shorthand for <a href="#">read()</a> <a href="#">More...</a>
void	<a href="#">rise (Callback&lt; void()&gt; func)</a>
	Attach a function to call when a rising edge occurs on the input. <a href="#">More...</a>
void	<a href="#">fall (Callback&lt; void()&gt; func)</a>
	Attach a function to call when a falling edge occurs on the input. <a href="#">More...</a>

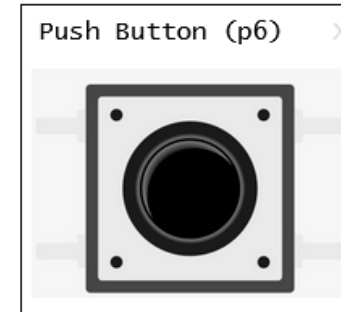
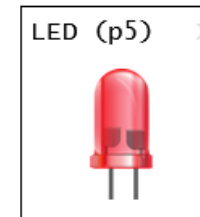
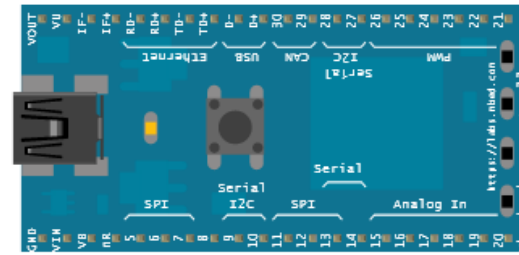
### 8.7.2 Interrupt sources

Each peripheral device has one interrupt line connected to the NVIC but may have several interrupt flags. Individual interrupt flags may also represent more than one interrupt source.

Any pin on Port 0 and Port 2 (total of 42 pins) regardless of the selected function, can be programmed to generate an interrupt on a rising edge, a falling edge, or both.

# Interrupciones externas

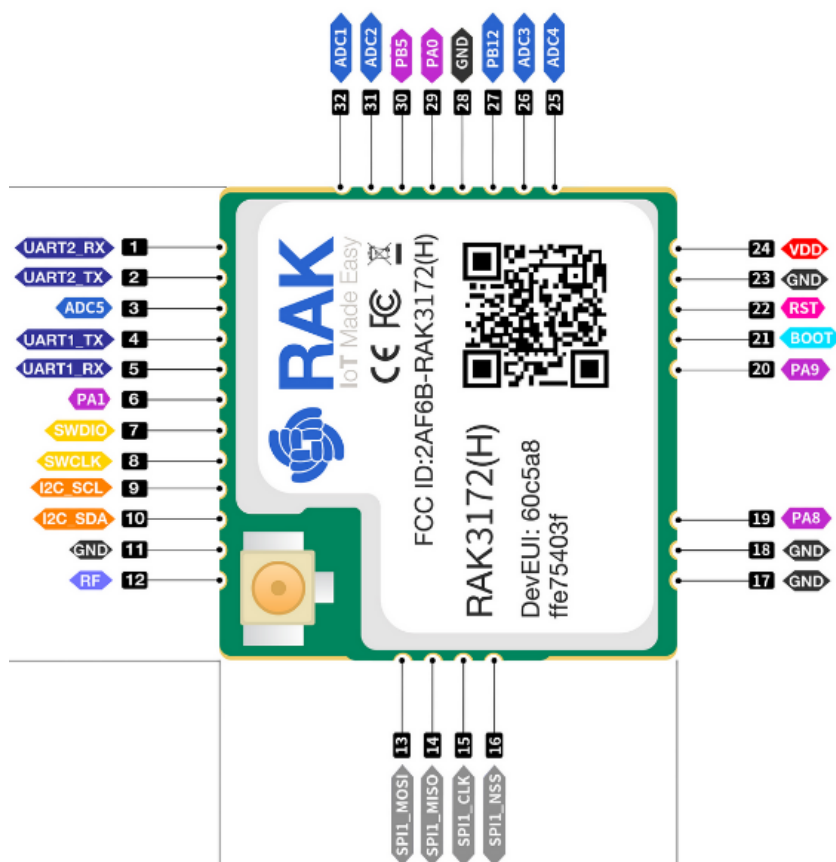
```
1 #include "mbed.h"
2
3 DigitalOut miLED(p5);
4 InterruptIn miBoton(p6);
5
6
7 void alternarLED() {
8     printf("LED alternado\n");
9     miLED.write(!miLED.read());
10 }
11
12
13
14 int main() {
15     miBoton.fall(&alternarLED);
16
17     while(1){
18         wait(1);
19     }
20 }
21
```



Serial output

LED alternado

# RAK3172



## Features

- Based on STM32WLE5CCU6
- LoRaWAN 1.0.3 specification compliant
- Supported bands: EU433, CN470, IN865, EU868, AU915, US915, KR920, RU864, and AS923-1/2/3/4
- LoRaWAN Activation by OTAA/ABP
- LoRa Point to Point (P2P) communication
- Easy to use AT Command Set via UART interface
- Long-range - greater than 15 km with optimized antenna
- Arm Cortex-M4 32-bit
- 256 kbytes flash memory with ECC
- 64 kbytes RAM
- Ultra-Low Power Consumption of 1.69  $\mu$ A in sleep mode
- Supply Voltage: 2.0 V ~ 3.6 V
- Temperature Range: -40° C ~ 85° C

# RAK3272

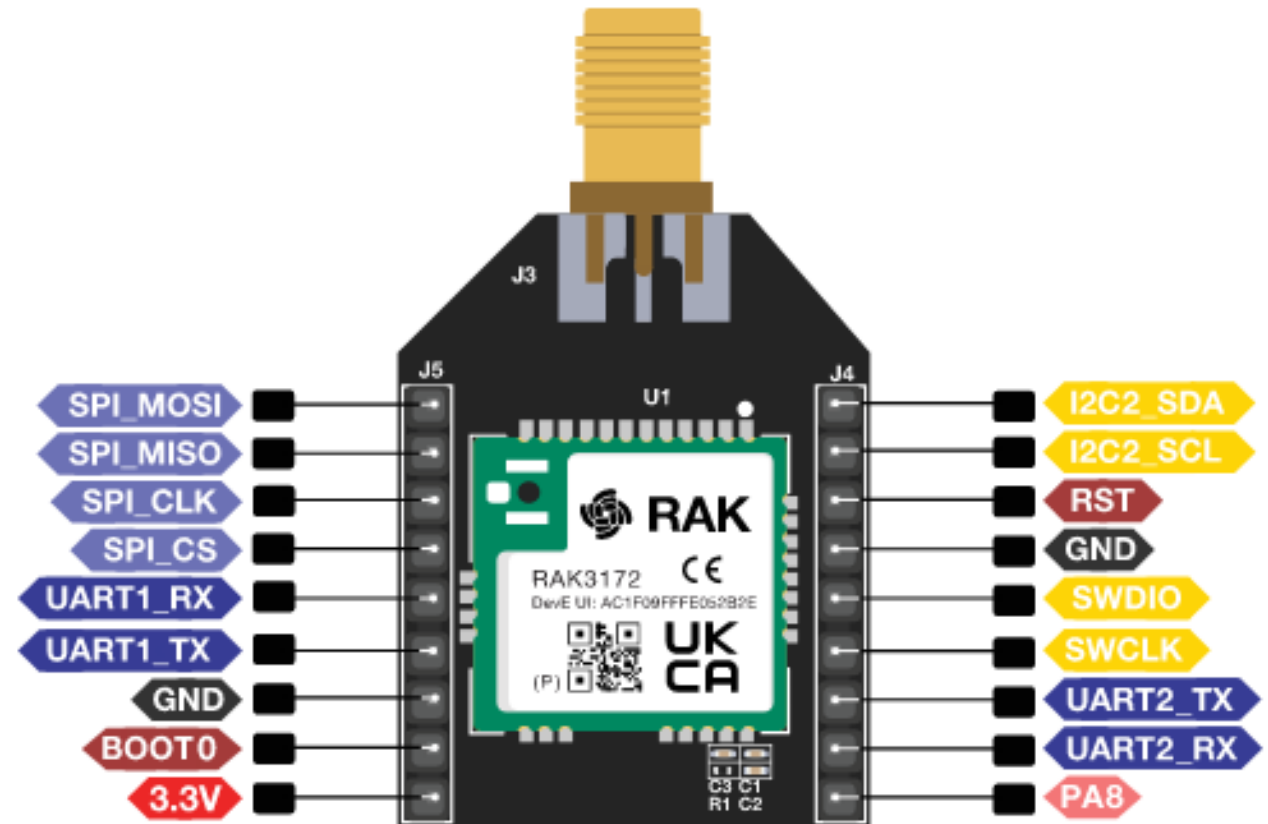
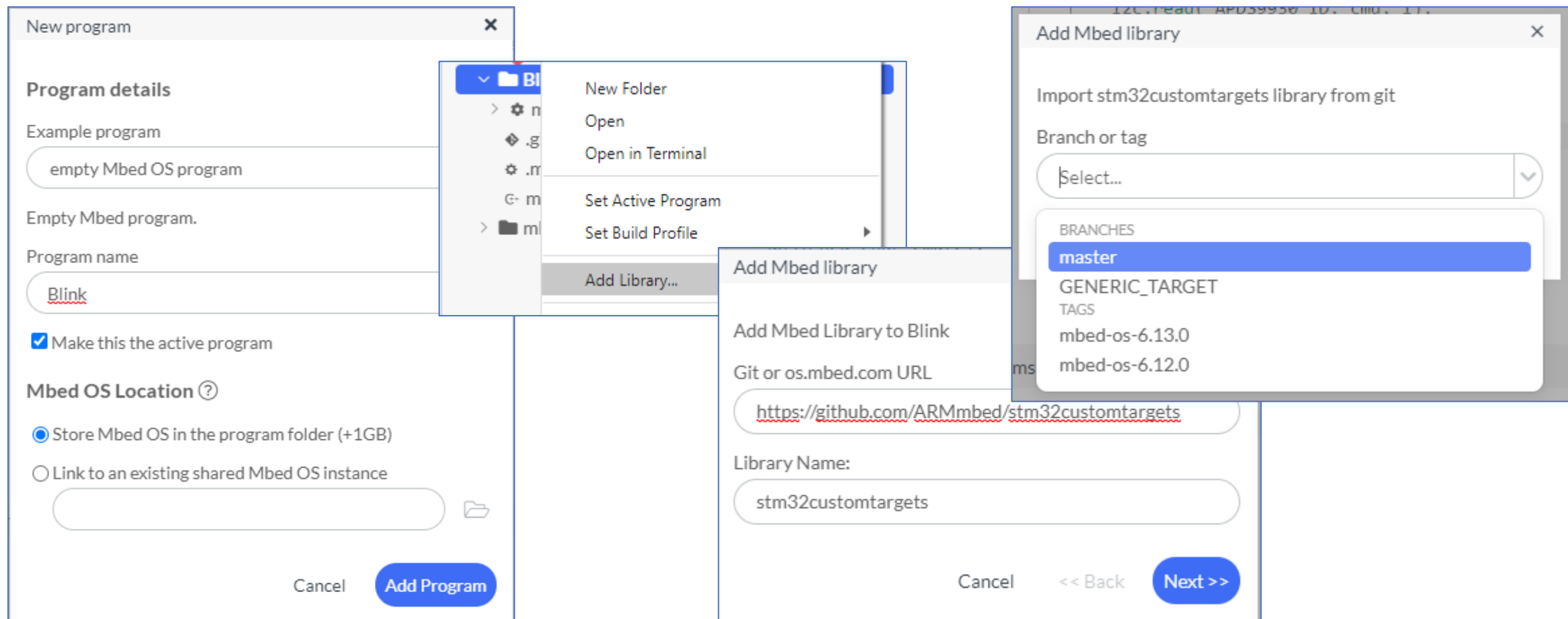


Figure 2: RAK3272S Breakout Board Pinout

# RAK3172 en Mbed Studio 1/3

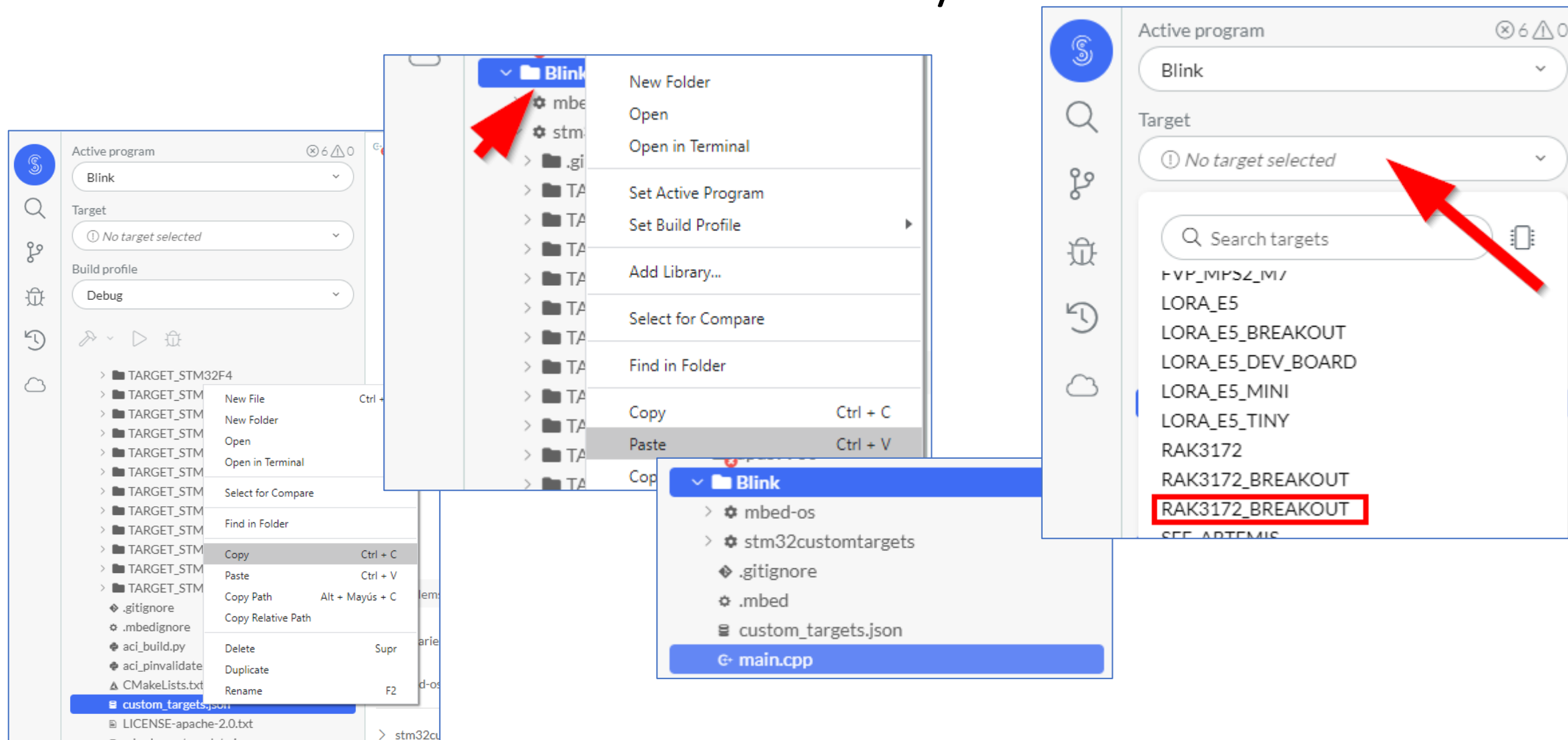
- Mbed Studio no incluye aún el módulo RAK3172
- Afortunadamente Charles Hallard ha incluido una placa basada en el RAK3172 en la librería de "Custom Targets" de Mbed
  - <https://github.com/ARMmbed/stm32customtargets>
  - Procedimiento:
    1. Crear un programa nuevo
    2. Importar la librería stm32customtargets
    3. Copiar el archivo custom\_targets.json de la librería anterior a la carpeta raíz del proyecto
    4. Seleccionar el nuevo target RAK3172\_BREAKOUT

# RAK3172 en Mbed Studio 2/3





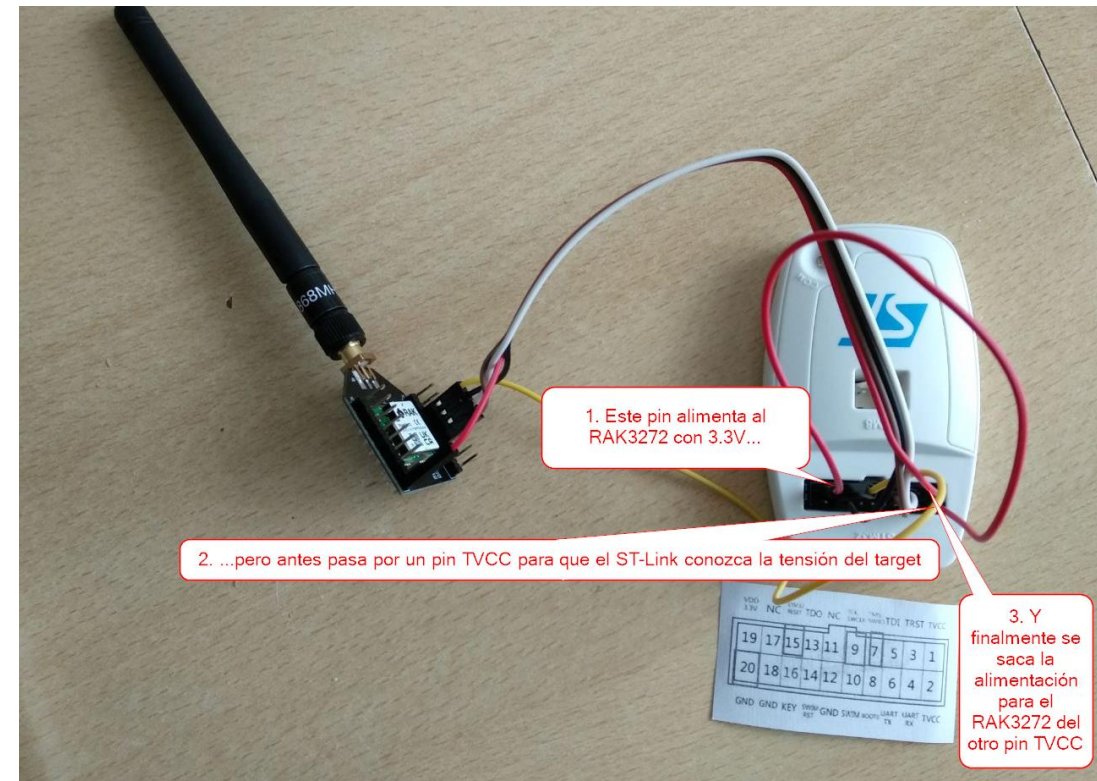
# RAK3172 en Mbed Studio 3/3



# Conexión del ST-Link v2

Se utiliza el pin 3.3V del ST-Link para alimentar el RAK3272, pero además tiene que conectarse a los pines TVCC para que el ST-Link "sepa" cuál es la tensión de alimentación del target.

Para poder programar el RAK3172 por primera es posible que tengamos que borrar previamente el firmware de comandos AT que trae de fábrica; esto podemos hacerlo con STM32CubeProgrammer.

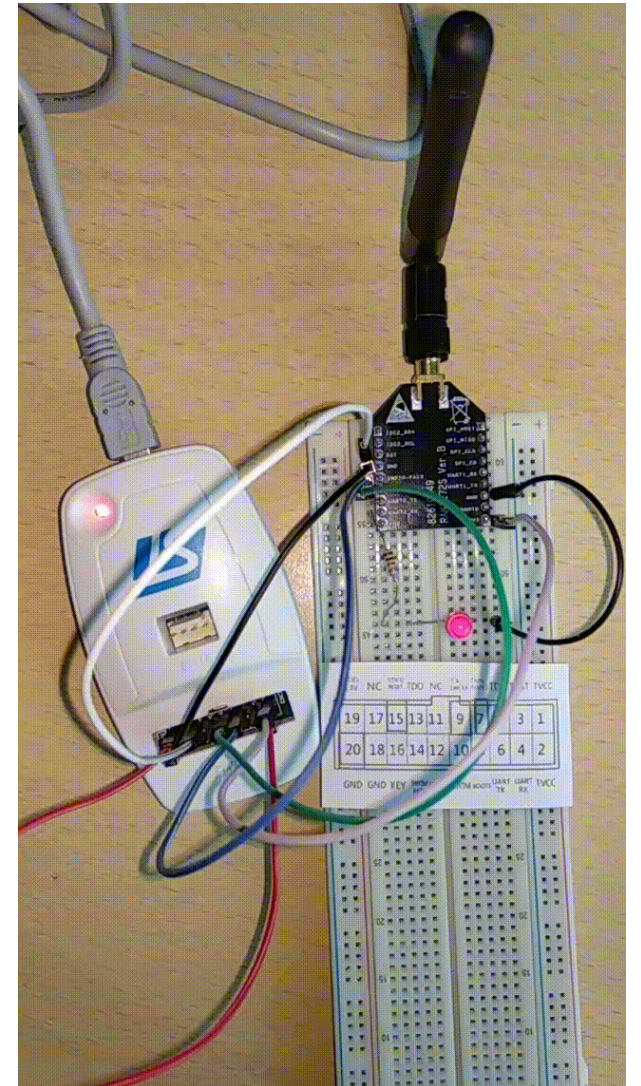


# Blink en el RAK3272

```
4  int main()
5  {
6      while (true) {
7          miLED.write(1);
8          //wait() ha sido deprecado en Mbed OS6
9          ThisThread::sleep_for(500ms);
10         miLED.write(0);
11         ThisThread::sleep_for(500ms);
12     }
13 }
14
```

Para que printf funcione por defecto, es necesario estar en el Build profile Debug; no Release.

[juanfelixmateos@gmail.com](mailto:juanfelixmateos@gmail.com)



# RTOS: Conceptos básicos

Thread

SysTick timer → RTOS ticker

Scheduler → Quantum time

Delays

Signals o EventFlags

Mutex

Semaphore

Queue y Memory pool

EventQueue

# Threads o Tasks: Hilos o tareas

---

Un proyecto puede requerir diversas tareas o threads.

Por ejemplo, en un reproductor de música podríamos definir 2 tareas:

- Atender la interfaz de usuario (botones para cambiar de canción, y pantalla para mostrar la información de la canción que se está reproduciendo).
- Reproducir el audio de la canción seleccionada.

No podemos esperar a que una tarea termine para ejecutar la otra; ambas tienen que ejecutarse **concurrentemente**.

Pero los microcontroladores de un único núcleo sólo pueden ejecutar una tarea a la vez, por lo que se utiliza un gestor (**Scheduler**) para asignar intervalos de ejecución (**time quantum**) a cada tarea y saltar de una a otra, creando la "ilusión" de que el sistema es capaz de ejecutar varias tareas simultáneamente (**multitasking**).

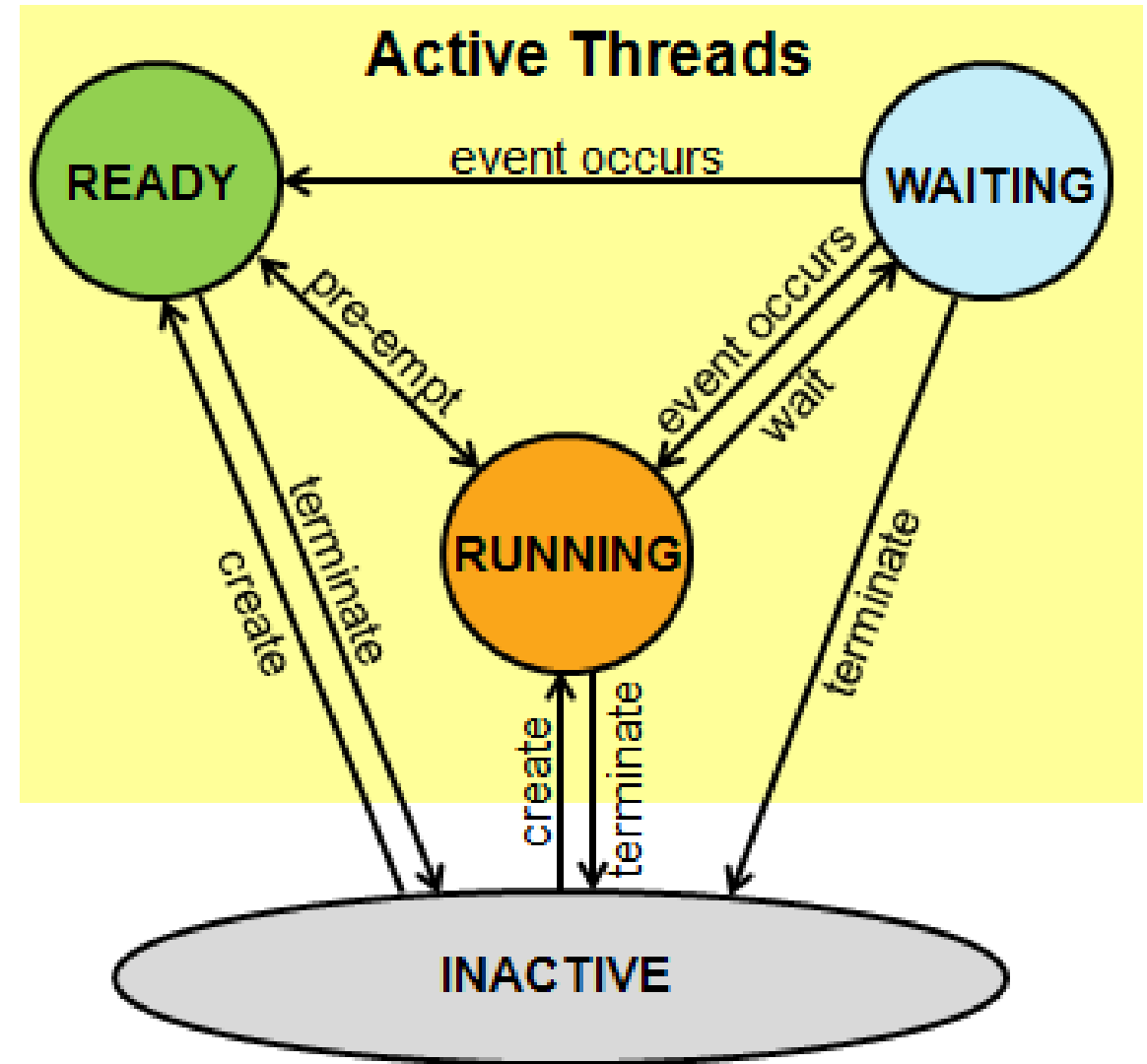
Mbed OS utiliza intervalos de ejecución de **1 ms** de duración.

La rutina **main** es un thread.

Además, existe un thread llamado **idle** que se ejecuta cuando no hay ningún thread en ejecución o están todos detenidos.

# Estados de un thread

- **RUNNING:** Es el thread que se está ejecutando. Sólo puede haber un thread en este estado.
- **READY:** Son threads que están esperando que el Scheduler les otorgue un turno de ejecución.
- **WAITING:** Threads que están esperando que se produzca un evento; cuando se produzca pasarán a estado READY.
- **INACTIVE:** Threads que aún no se han iniciado o han terminado.





# Priority based round robin Scheduler

Mbed utiliza (por defecto) un Scheduler de tipo **Round Robin**, que se caracteriza por ir saltando de un thread a otro dedicando exactamente el mismo tiempo de ejecución a cada uno de ellos, pero...

...después de cada time quantum, elige a qué thread saltar en función de la prioridad con la que hayan sido definidos.

Si hay un thread con más prioridad que los demás, no consentirá que la ejecución pase a otro thread hasta que él termine o pase a modo wait.

PROCESS	ARRIVAL TIME	BURST TIME		PRIORITY
		TOTAL	REMAINING	
P1	0	4	4	4
P2	1	3	3	3
P3	3	4	4	1
P4	6	2	2	5
P5	8	4	4	2

GANTT CHART:



# Modos de ejecución: Thread o Handler

El microcontrolador puede adoptar 2 modos de funcionamiento:

- **Handler o Interrupt:** Es un modo privilegiado en el que el código tiene acceso a todos los recursos del microcontrolador, y que se activa en respuesta a un evento, como puede ser el cambio de estado de un pin (o eventos internos que puede generar el propio RTOS). Generalmente se usa este modo cuando necesitamos responder lo más rápido posible al evento (reducir la **latencia** al mínimo). En general, pero especialmente al usar un RTOS, es esencial que el tiempo que el microcontrolador pasa en modo Handler sea el mínimo posible, porque durante este periodo el RTOS no puede atender otras tareas, pudiendo incluso **comprometer su funcionamiento**.
- **Thread o segundo plano:** El código que se ejecuta en este modo no requiere tanta "velocidad de reacción". La mayoría del código se ejecuta en este modo. De hecho el código que se ejecuta en modo Handler, que se denomina **Interrupt Service Routine (ISR)** generalmente sólo activa una bandera (flag) para que posteriormente un thread se encargue de realizar las tareas de respuesta a ese evento.



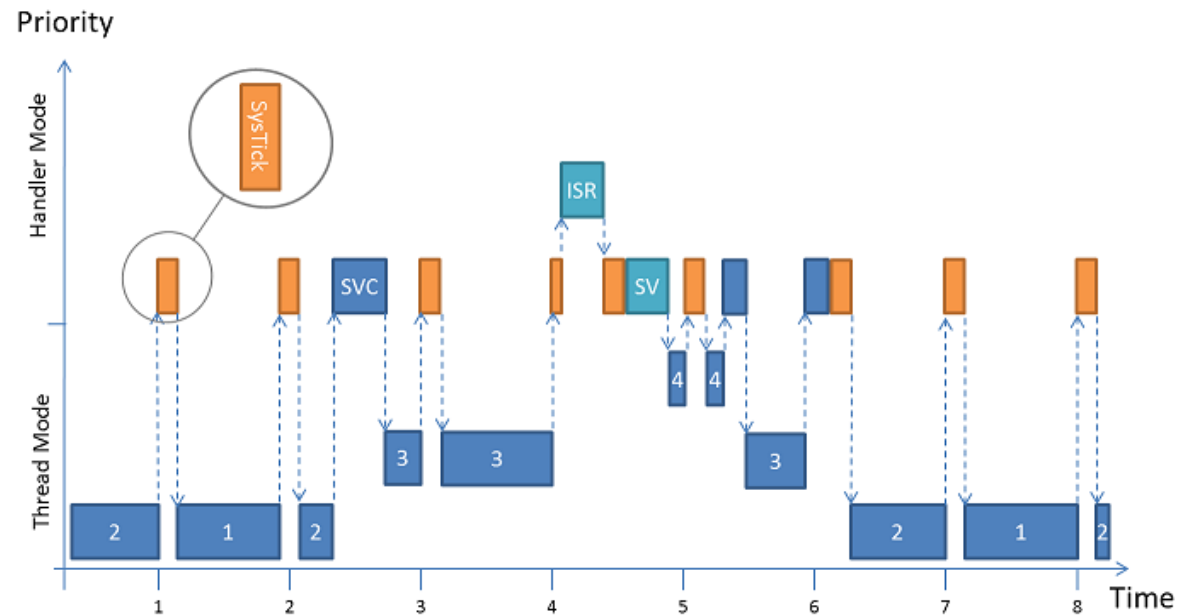
# El difícil equilibrio entre el RTOS y las interrupciones

Si damos prioridad a las interrupciones, el RTOS puede quedar comprometido, porque hay operaciones mínimas que tiene que realizar periódicamente.

Si damos prioridad a los thread, las interrupciones aumentarán su latencia.

¿Solución?

El RTOS ejecuta en cada quantum las tareas mínimas de mantenimiento, luego cede el control a las ISR si las hubiera, y luego realiza el resto de sus tareas. Las ISR deben ser cortas, para no superar el tiempo de quantum.



# Diferencias entre Threads y funciones

- Los Thread siempre contienen un bucle infinito
- Las funciones siempre devuelven un valor

```
unsigned int procedure (void)
{
    .....
    return(ch);
}
```

```
void thread (void)
{
    while(1)
    {
        .....
    }
}
```

# Thread & ThisThread

## Thread

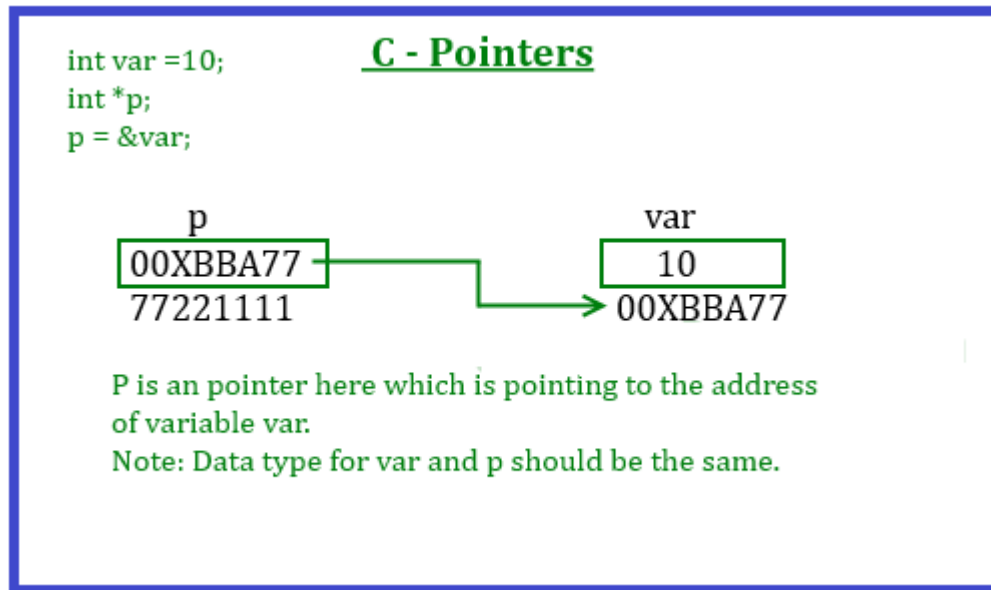
- Constructor: Thread (prioridad)
- start(nombre) → Inicia el Thread
- start(callback(nombre,argumentos))
- get\_state()
- get\_id()
- terminate()

ThisThread es una **espacio de nombres**, no una clase (no se pueden crear objetos suyos)

```
#include "mbed.h"
DigitalOut miLED(PA_8);
Thread thread;
void led_thread(){
    while (true) {
        miLED = !miLED;
        printf("Toggle LED!\r\n");
        ThisThread::sleep_for(1s);
    }
}
int main(){
    printf("ID main: %d\r\n",ThisThread::get_id());
    thread.start(led_thread);
    printf("ID led_thread: %d\r\n",thread.get_id());
    printf("State: %d\r\n",thread.get_state());
    ThisThread::sleep_for(10s);
    thread.terminate();
    printf("State: %d\r\n",thread.get_state());
}
```

# Thread Callback

Para pasar argumentos a un thread tenemos que usar un callback



```
#include "mbed.h"
DigitalOut miLED(PA_8);
Thread thread;
void led_thread(DigitalOut *led){
    while (true) {
        *led = !*led;
        printf("Toggle LED!\r\n");
        ThisThread::sleep_for(1s);
    }
}
int main(){
    printf("ID main: %d\r\n",ThisThread::get_id());
    thread.start(callback(led_thread,&miLED));
    printf("ID led_thread: %d\r\n",thread.get_id());
    printf("State: %d\r\n",thread.get_state());
    ThisThread::sleep_for(10s);
    thread.terminate();
    printf("State: %d\r\n",thread.get_state());
}
```

# ¿Qué ocurrirá en este programa con 2 threads?

```
#include "mbed.h"

Thread alternar_led;
DigitalOut led1(PA_8);
volatile bool running = true;
void blink(DigitalOut *led) {
    while (true){
        if(running) {
            *led = 1;
            ThisThread::sleep_for(2s);
            *led = 0;
            ThisThread::sleep_for(2s);
        }
    }
}
```

```
int main() {
    alternar_led.start(callback(blink, &led1));
    while(true){
        ThisThread::sleep_for(10s);
        running = false;
        ThisThread::sleep_for(6s);
        running = true;
    }
}
```

1. Se ejecuta el thread main, que a su vez inicia la ejecución del thread `alternar_led` y, a continuación, entra en un bucle infinito en el que alterna el valor de la variable booleana `running` a intervalos de 10 y 6 segundos.
2. El thread `alternar_led`, si la variable `running` es `true`, hace parpadear un LED una vez cada 4 segundos.
3. El LED se encenderá 3 veces durante 2 segundos, seguidos de 2 segundos de apagado, salvo la tercera vez, que el apagado durará 6 segundos... y se continuará ejecutando este patrón indefinidamente.

# Conceptos clave: Concurrencia y sincronización

Para que los threads puedan "cooperar" entre sí, no podemos encomendárselo todo a las esperas (`sleep_for`), necesitamos que puedan intercambiar información entre sí.



# EventFlags: Señalización entre threads 1/2

Las EventFlags nos permiten detener la ejecución de cualquier hilo hasta que se activen ciertas banderas (flags).

Una bandera es simplemente un bit, que estará activado cuando adquiera el valor 1.

Cada EventFlags puede contener hasta 31 banderas.

# EventFlags: Señalización entre threads 2/2

```
#include "mbed.h"

#define SAMPLE_FLAG1 (1UL << 0)
#define SAMPLE_FLAG2 (1UL << 9)

EventFlags event_flags;

void worker_thread_fun(){
    printf("Waiting for any flag from 0x%08lx.\n", SAMPLE_FLAG1 | SAMPLE_FLAG2);
    uint32_t flags_read = 0;
    while (true) {
        flags_read = event_flags.wait_any(SAMPLE_FLAG1 | SAMPLE_FLAG2);
        printf("Got: 0x%08lx\n", flags_read);
    }
}

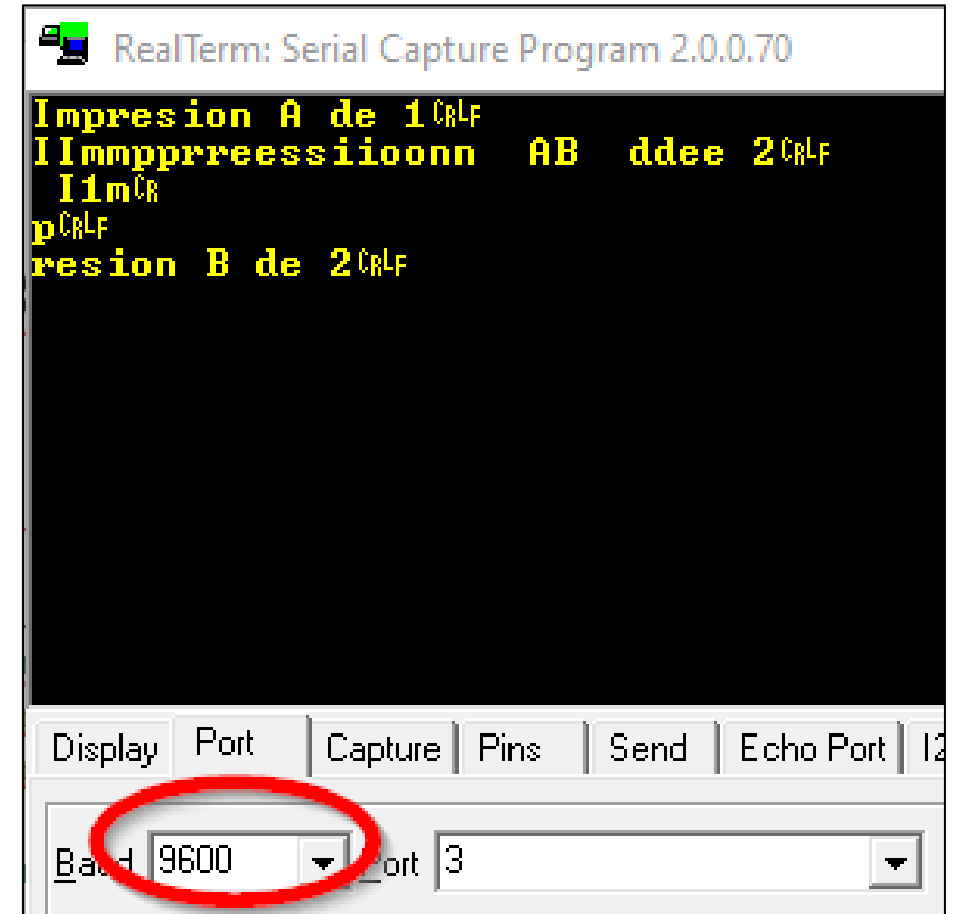
int main(){
    Thread worker_thread;
    worker_thread.start(mbed::callback(worker_thread_fun));
    while (true) {
        ThisThread::sleep_for(1000);
        event_flags.set(SAMPLE_FLAG1);
        ThisThread::sleep_for(500);
        event_flags.set(SAMPLE_FLAG2);
    }
}
```



# Mutex: Mutual Exclusion 1/2

```
#include "mbed.h"

Thread thread1, thread2;
void doble_impresion (int i){
    printf("Impresion A de %d\r\n",i);
    printf("Impresion B de %d\r\n",i);
}
void codigo_thread1(){
    doble_impresion(1);
}
void codigo_thread2(){
    doble_impresion(2);
}
int main() {
    thread1.start(codigo_thread1);
    thread2.start(codigo_thread2);
    thread1.join();
    thread2.join();
}
```



# Mutex: Mutual Exclusion 2/2

```
#include "mbed.h"

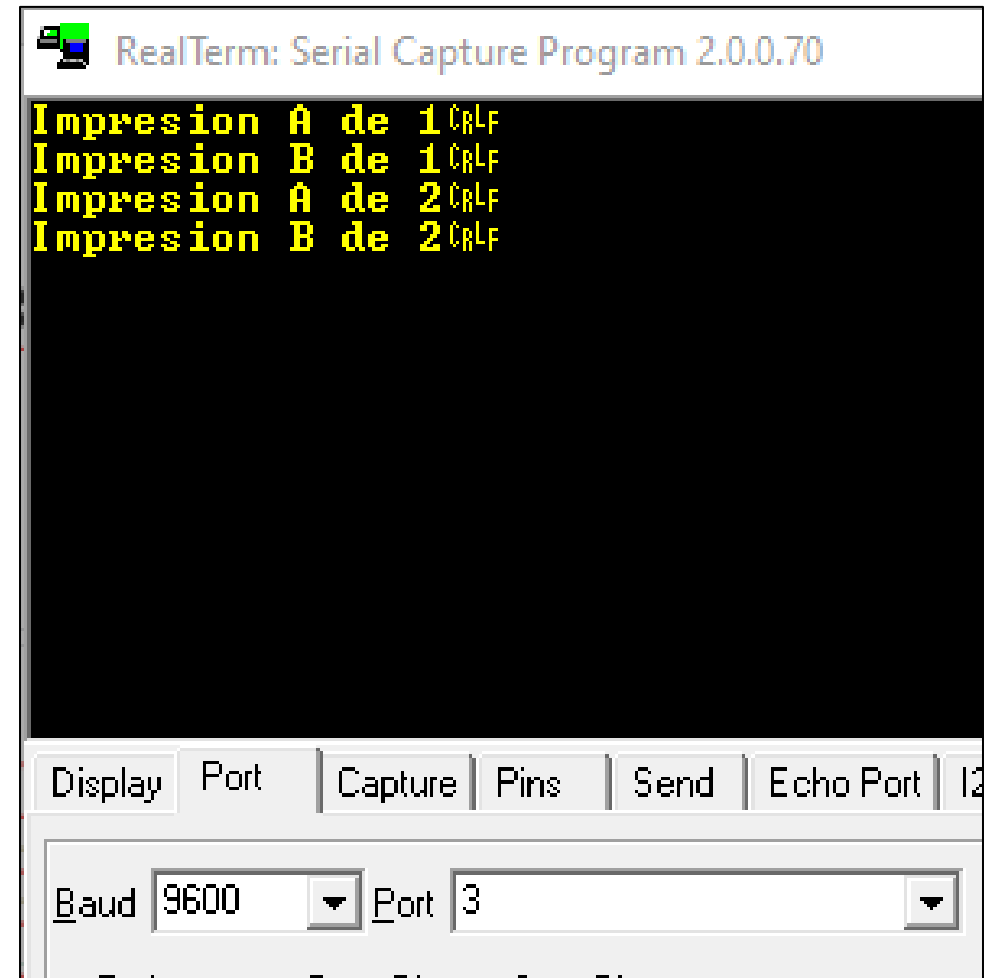
Thread thread1, thread2;
Mutex compartir;

void doble_impresion (int i){
    compartir.lock();
    printf("Impresion A de %d\r\n",i);
    printf("Impresion B de %d\r\n",i);
    compartir.unlock();
}

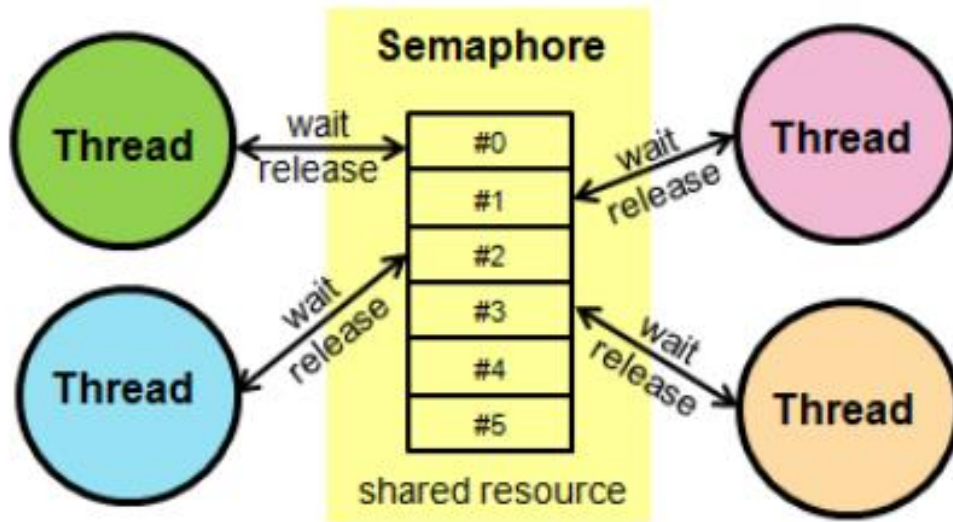
void codigo_thread1(){
    doble_impresion(1);
}

void codigo_thread2(){
    doble_impresion(2);
}

int main() {
    thread1.start(codigo_thread1);
    thread2.start(codigo_thread2);
    thread1.join();
    thread2.join();
}
```



# Semaphore 1/3



- Un semáforo es un **repartidor de tokens**, al que los threads le pueden solicitar tokens (acquire) o añadirselos (reléase).
- Si un thread desea adquirir un token, pero no hay ninguno disponible en el semáforo, tendrá que esperar.

# Semaphore 2/3

```
#include "mbed.h"

Semaphore continuar(0);
DigitalOut led(PA_8);
Thread t1;
Thread t2;

void codigo_t1(){
    while (true) {
        printf("Voy a dejar que t2 parpadee el LED una vez dentro de 10 segundos\r\n");
        ThisThread::sleep_for(10s);
        continuar.release();
    }
}
```

```
void codigo_t2(){
    while (true) {
        continuar.acquire();
        led=1;
        ThisThread::sleep_for(500ms);
        led=0;
        ThisThread::sleep_for(500ms);
    }
}

int main(void){
    t1.start(codigo_t1);
    t2.start(codigo_t2);
}
```

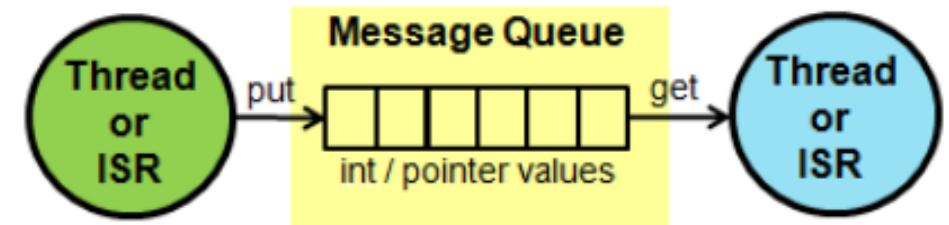
# Semaphore 3/3

Cuándo utilizar semáforos:

- Enviar **señales** entre threads (diapositiva anterior)
- **Multiplexar** el acceso de varios threads a un conjunto limitado de recursos (por ejemplo, a una memoria con doble puerto)
- **Redezvous**: Es un caso particular de envío de señales en el que 2 threads se envían mensajes entre sí al alcanzar cierto punto crítico, para garantizar que ninguno de ellos sigue adelante sin que el otro haya alcanzado también su punto crítico.
- **Barrera**: Es un caso más general de Redezvous que sirve para cualquier número de threads (ninguno continuará hasta que todos hayan alcanzado su punto crítico). Se usan 2 semáforos (contador y barrera) y una variable global (terminado). Inicialmente contador tiene un token y barrera ninguno. Cada vez que un hilo alcanza su punto crítico, intenta adquirir un token de contador, incrementa el valor de terminado, libera el token de contador para que lo pueda usar otro thread, comprueba si terminado es igual al número total de threads, en cuyo caso añade tanto tokens a barrera como threads, o en caso contrario espera hasta poder adquirir un token de barrera.

# Queues

- Los métodos de sincronización anteriores (eventFlags, mutex y semaphores) sólo permite iniciar la ejecución de threads, pero no **intercambiar información** entre ellos.
- Para intercambiar información, podemos usar la clase Queue, que sirve para pasar **punteros** de un thread a otro
- El thread **productor** inserta punteros en la cola (queue), indicando su **prioridad**, y el thread consumidor los extraerá de ella por orden de prioridad, o si todos tienen la misma, por FIFO.



# EventQueues 1/3

Un Event Queue, o cola de eventos, sirve para que podamos solicitar la ejecución pautada (con demoras y/o periodicidad) de funciones.

Suele usarse para sincronizar varios threads, o para diferir la ejecución del código de una interrupción al modo thread.

Por ejemplo, podemos solicitar que se envíe un mensaje LoRaWAN cada hora, o en la ISR de una interrupción podemos solicitar que se "ponga en cola" el código de negocio de la interrupción (evitando así que el procesador esté en modo interrupción más tiempo del imprescindible).

Al declarar la cola tendremos que asignarle un tamaño (número de eventos que puede contener).

Podemos ejecutar la cola para que despache los eventos continuamente, durante cierto periodo de tiempo, o hasta que termine con todos los eventos que estén actualmente en la cola.

# EventQueues 2/3

```
#include "mbed.h"
```

```
// Create a queue that can hold a maximum of 32 events
```

```
EventQueue queue(32 * EVENTS_EVENT_SIZE);
```

```
// Create a thread that'll run the event queue's dispatch function
```

```
Thread t;
```

```
int main () {
```

```
    // Start the event queue's dispatch thread
```

```
    t.start(callback(&queue, &EventQueue::dispatch_forever));
```

```
}
```



# EventQueues 3/3: Diferir una interrupción

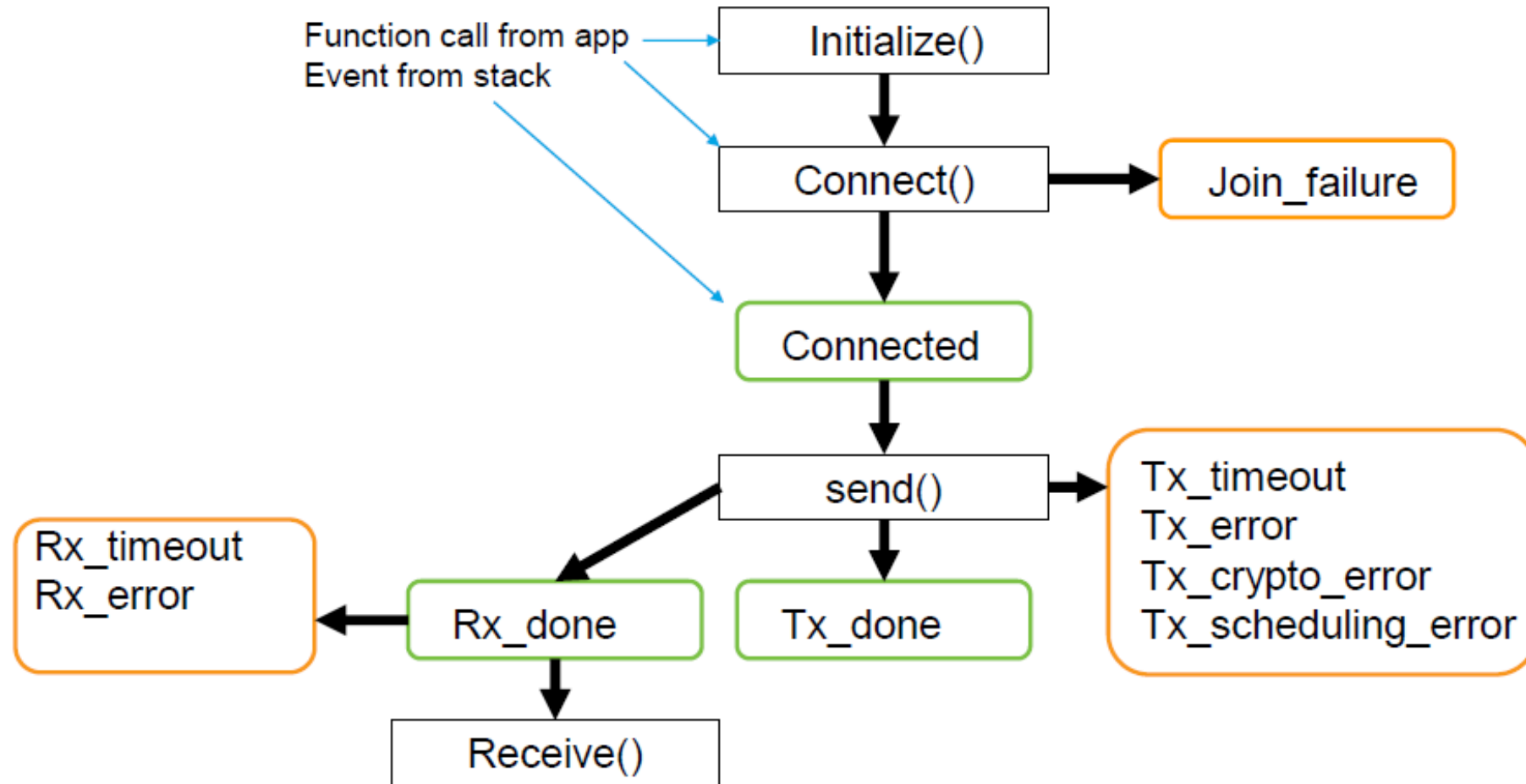
```
#include "mbed.h"

DigitalOut led1(LED1);
InterruptIn sw(SW2);
EventQueue queue(32 * EVENTS_EVENT_SIZE);
Thread t;

void fall_handler(void){
    printf("rise_handler in context %p\n", ThisThread::get_id());
    // Toggle LED
    led1 = !led1;
}

int main(){
    t.start(callback(&queue, &EventQueue::dispatch_forever));
    sw.fall(queue.event(fall_handler));
}
```

# LoRa in Mbed – Example stack events visible to application



# LoRaWAN API → 3 elementos fundamentales (1/2)

- **EventQueue** → Se la pasamos al método initialize del stack LoRaWAN. En esta cola introducimos las órdenes de enviar mensajes (bien periódicas, o bien puntuales en respuesta a interrupciones)
- **lorawan\_app\_callbacks** → En su propiedad events tenemos que almacenar una referencia callback al LoRa event handler. Actúa como nexo de unión entre el stack LoRaWAN y el código (LoRa evento handler) encargado de gestionar los eventos.
- **LoRa event handler** → Es el switch en el que realmente se procesan los eventos del stack LoRaWAN (CONNECTED, TX\_DONE, UPLINK\_REQUIRED...)

# LoRaWAN API → 3 elementos fundamentales (2/2)

- **EventQueue**

- `ev_queue.call_every(30000, send_message);`

- **lorawan\_app\_callbacks**

- Se encarga de enviar cualquier evento que genere el stack LoRaWAN (por ejemplo, un TX\_DONE) al LoRa evento handler.

- **LoRa event handler**

- Procesa los eventos del stack LoRaWAN; por ejemplo:

- case DISCONNECTED:

- `ev_queue.break_dispatch();`

- `printf("\r\n Disconnected Successfully \r\n");`

- `break;`

# Procedimiento general para crear una aplicación LoRaWAN

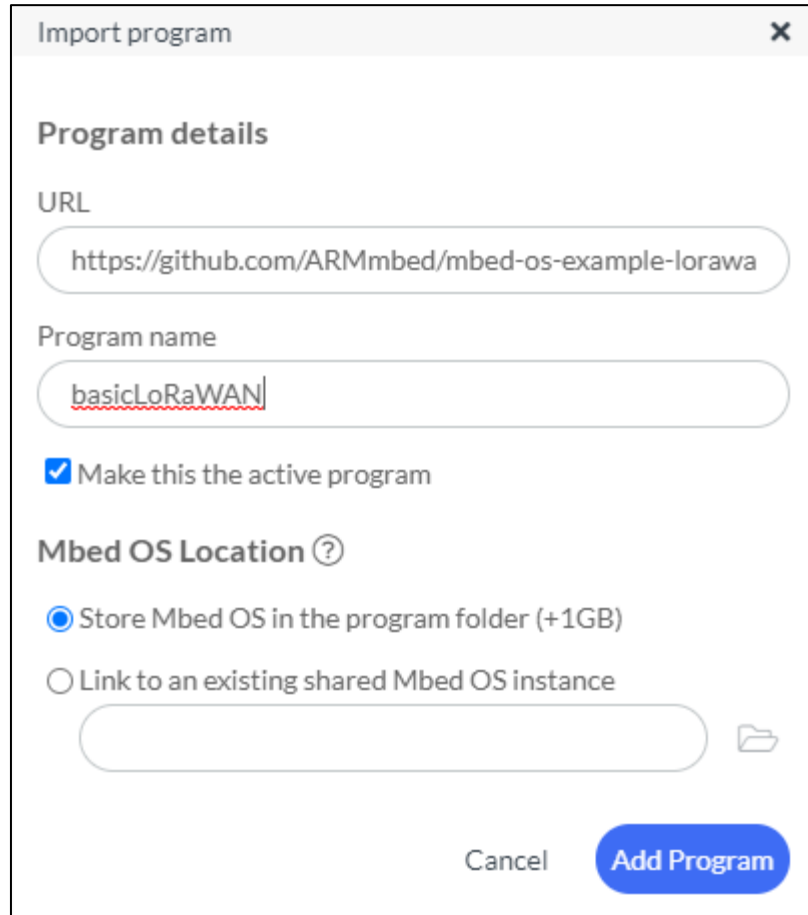
- Core parts of application
  - Define variables.
  - Create and initialize radio instance with correct IO-pins.
  - Create LoRaWAN interface object.
  - Create eventQueue object.
    - Add your callbacks to eventQueue
  - Create state machine.
    - Minimum states needed: wait, tx\_done, read\_temperature, send, tx\_timeout, rx\_done.
  - Create event handler function.
    - Minimum events to be handled: connected, tx\_done, tx\_timeout, rx\_done.
  - Set keys and connection type.
  - Initialize and connect your LoRaWAN node.

Afortunadamente existe un código de ejemplo que nos facilita todas estas tareas

# Mbed OS + LoRaWAN + RAK3172

1. Importar el programa de ejemplo
  - <https://github.com/ARMmbed/mbed-os-example-lorawan>
2. Importar la librería `stm32customtargets`
  - <https://github.com/ARMmbed/stm32customtargets>
3. Copiar el contenido del archivo `custom_targets.json` de la librería anterior y pegarlo reemplazando al que hay en la carpeta raíz del proyecto
4. Seleccionar el nuevo target `RAK3172_BREAKOUT`
5. Añadir la configuración del `RAK3172_BREAKOUT` en el archivo `mbed_app.json`
6. Configurar los datos del nodo en el archivo `mbed_app.json`

# 1. Importar el programa de ejemplo



Import program

**Program details**

URL

<https://github.com/ARMmbed/mbed-os-example-lorawa>

Program name

basicLoRaWAN

☒ Make this the active program

**Mbed OS Location** ?

☒ Store Mbed OS in the program folder (+1GB)

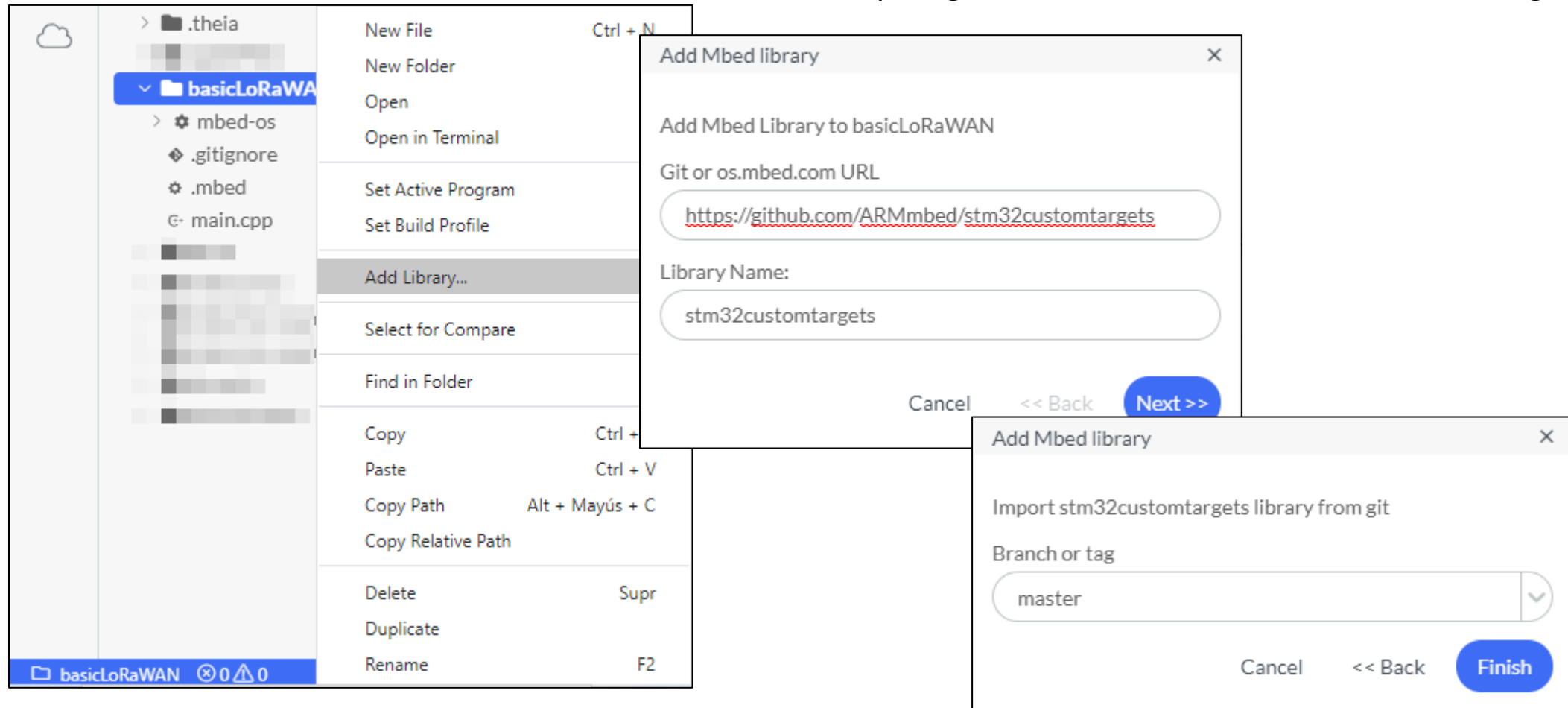
☐ Link to an existing shared Mbed OS instance

Cancel Add Program

<https://github.com/ARMmbed/mbed-os-example-lorawan.git>

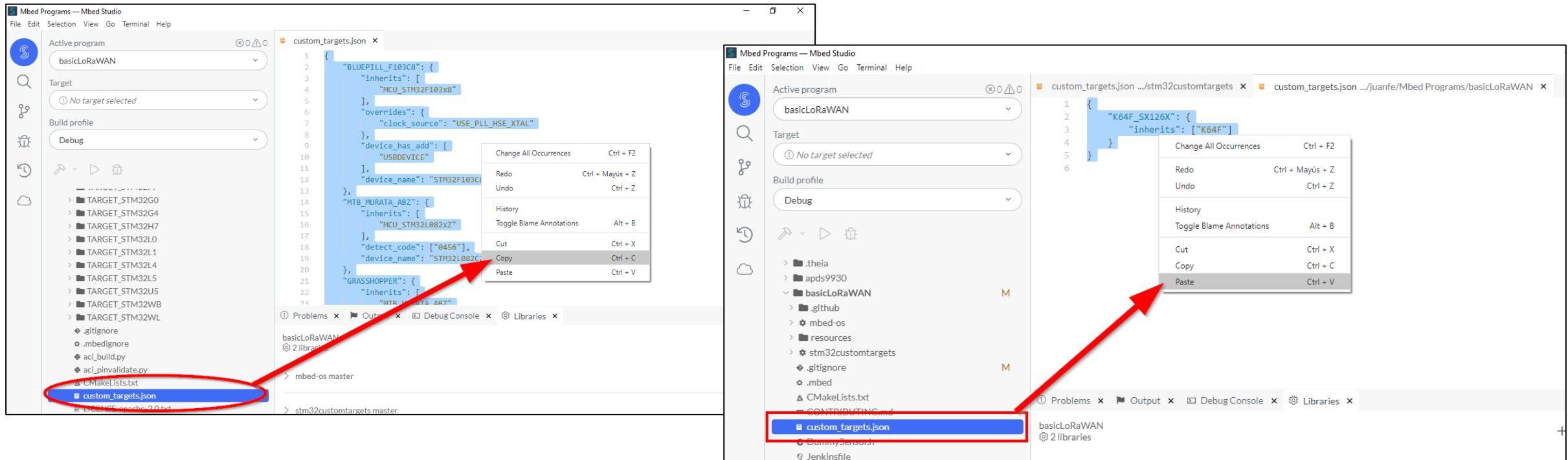
## 2. Importar la librería

<https://github.com/ARMmbed/stm32customtargets>

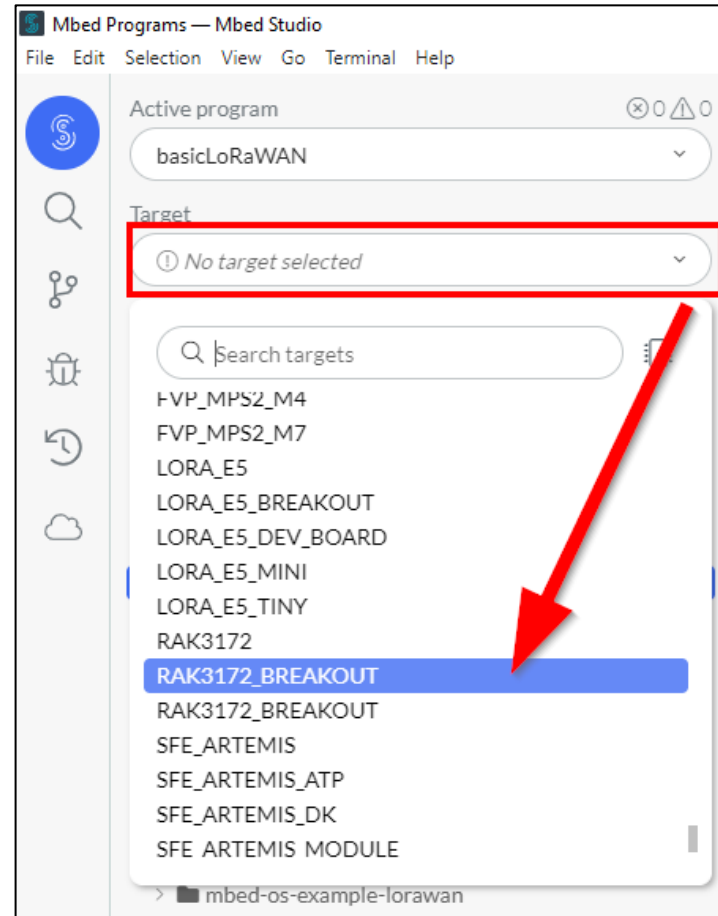




### 3. Copiar el contenido del archivo custom\_targets.json de la librería anterior y pegarlo reemplazando al que hay en la carpeta raíz del proyecto



## 4. Seleccionar el nuevo target RAK3172\_BREAKOUT



## 5. Añadir la configuración del RAK3172\_BREAKOUT en el archivo mbed\_app.json

```
200
201     "NUCLEO_L073RZ": {
202         "main_stack_size":      2048,
203         "target.components_add": ["SX126X"],
204         "SX126X-lora-driver.spi-mosi": "ARDUINO_UNO_D11",
205         "SX126X-lora-driver.spi-miso": "ARDUINO_UNO_D12",
206         "SX126X-lora-driver.spi-sclk": "ARDUINO_UNO_D13",
207         "SX126X-lora-driver.spi-cs":   "ARDUINO_UNO_D7",
208         "SX126X-lora-driver.reset":    "ARDUINO_UNO_A0",
209         "SX126X-lora-driver.dio1":     "ARDUINO_UNO_D5",
210         "SX126X-lora-driver.busy":     "ARDUINO_UNO_D3",
211         "SX126X-lora-driver.freq-select": "ARDUINO_UNO_A1",
212         "SX126X-lora-driver.device-select": "ARDUINO_UNO_A2",
213         "SX126X-lora-driver.crystal-select": "ARDUINO_UNO_A3",
214         "SX126X-lora-driver.ant-switch":  "ARDUINO_UNO_D8"
215     },
216     "RAK3172_BREAKOUT": {
217         "stm32wl-lora-driver.rf_switch_config": 2,
218         "stm32wl-lora-driver.crystal_select": 0
219     }
220 },
221 "macros": ["MBEDTLS_USER_CONFIG_FILE=\"mbedtls_lora_config.h\""]
222 }
223
224
```

## 6. Configurar los datos del nodo en el archivo mbed\_app.json

```
mbed_app.json x
1  [
2    "config": {
3      "main_stack_size": { "value": 4096 }
4    },
5    "target_overrides": {
6      {
7        "platform.stdio-convert-newlines": true,
8        "platform.stdio-baud-rate": 115200,
9        "platform.default-serial-baud-rate": 115200,
10       "mbed-trace.enable": false,
11       "mbed-trace.max-level": "TRACE_LEVEL_DEBUG",
12       "lora.over-the-air-activation": true,
13       "lora.duty-cycle-on": true,
14       "lora.phy": "EU868",
15       "lora.device-eui": "{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }",
16       "lora.application-eui": "{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }",
17       "lora.application-key": "{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }",
18     },
19  ]
```

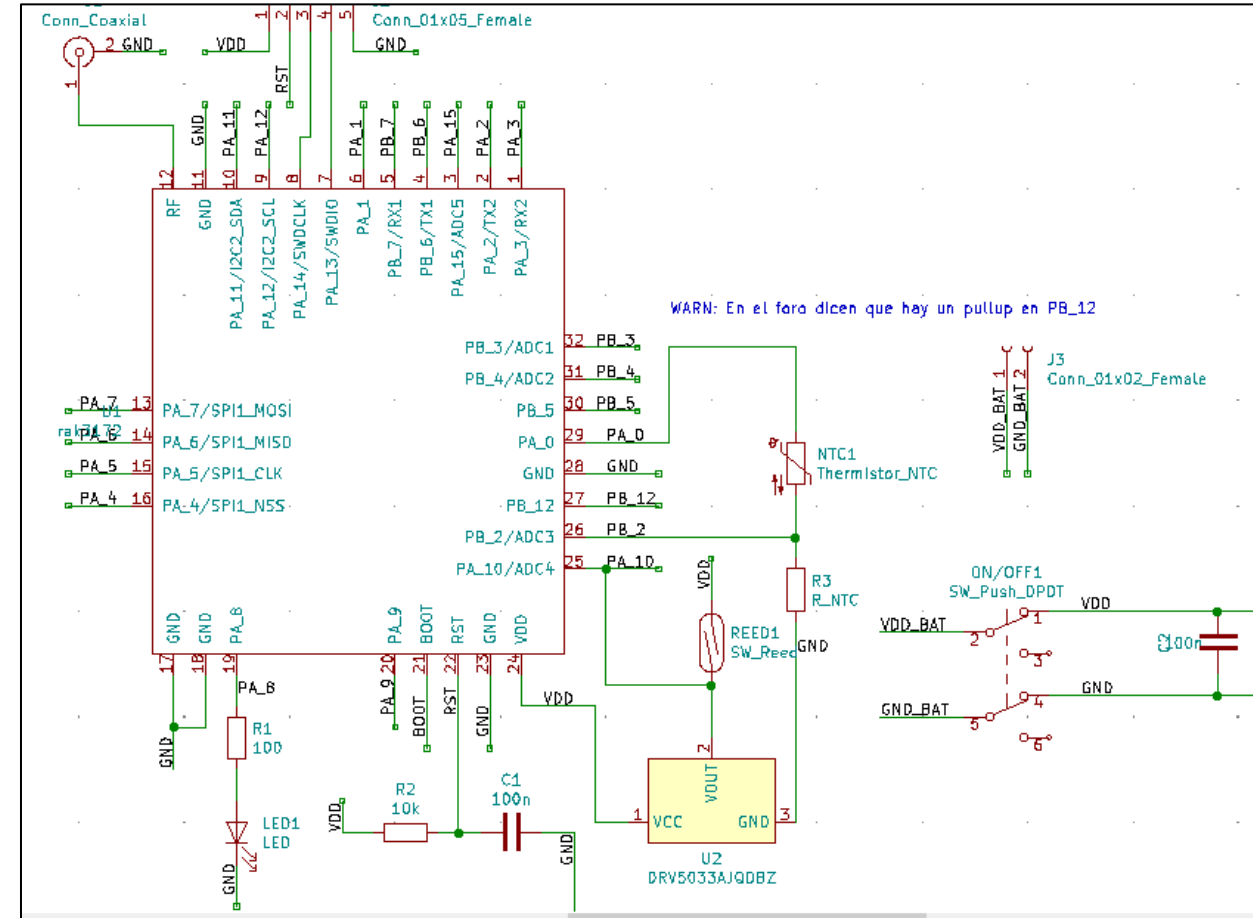
### Para ABP

```
"lora.over-the-air-activation": false,
"lora.appskey": "{ YOUR_APPLICATION_SESSION_KEY }",
"lora.nwkskey": "{ YOUR_NETWORK_SESSION_KEY }",
"lora.device-address": "
YOUR_DEVICE_ADDRESS_IN_HEX "
```

Copiar los datos de la consola de TTN en formato msb

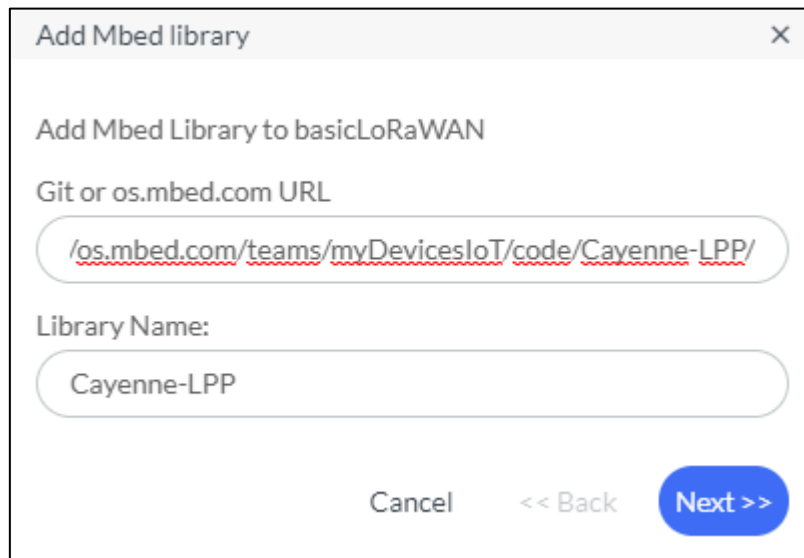
# Modificar el ejemplo para el TTNMAD\_DOOR\_RAK3172

- Usaremos 2 interrupciones:
  - Una de flanco de bajada para el sensor de puerta abierta
  - Otra de LowPowerTimeout para despertar del modo de bajo consumo a intervalos regulares y enviar un uplink de tipo heartbeat
- Usaremos el formato Cayenne LPP para enviar los uplinks
- El sensor de puerta abierta (hall o reed) es de tipo NC (en presencia del imán se cierra) y está conectado al pin PA\_10, en el que habilitaremos la resistencia pull-down interna.
- La NTC tiene una resistencia pull-up, se alimenta por el pin PA\_0 y se lee en el pin PB\_2.



# Importar la librería Cayenne LPP

- <https://os.mbed.com/teams/myDevicesIoT/code/Cayenne-LPP>
- Definimos la carga de pago lpp(11), las funciones para envío de los uplinks, y algunas variables



Add Mbed library

Add Mbed Library to basicLoRaWAN

Git or os.mbed.com URL

/os.mbed.com/teams/myDevicesIoT/code/Cayenne-LPP/

Library Name:

Cayenne-LPP

Cancel << Back Next >>

```
24 // Application helpers
25 // #include "DummySensor.h"
26 #include "trace_helper.h"
27 #include "lora radio helper.h"
28 #include "CayenneLPP.h"
29 CayenneLPP lpp(11);
30 bool puertaAbierta;
31 int periodoHeartbeat; //En minutos
32 bool uplinkACK;
33
34 using namespace events;
```

# Declarar las interrupciones y deshabilitar el ADR

```
61 | #define PC_9 0
62 |
63 | /**
64 |  * Dummy sensor class object
65 |  */
66 | // DS1820 ds1820(PC_9);
67 |
68 | InterruptIn reed(PA_10, PullDown);
69 | LowPowerTimeout heartbeat;
70 |
71 | /**
72 |  * This event queue is the global event queue
```

```
133 | /*
134 |  * Enable adaptive data rate
135 |  * if (lorawan.enable_adaptive_datarate() != LORAWAN_STATUS_OK) {
136 |  *     printf("\r\n enable_adaptive_datarate failed! \r\n");
137 |  *     return -1;
138 |  * }
139 |
140 |  * printf("\r\n Adaptive data rate (ADR) - Enabled \r\n");
141 |  * lorawan.disable_adaptive_datarate();
142 |  *
143 |  * retcode = lorawan.connect();
144 |
145 |  */
```

# Enviar un primer uplink al iniciar el sistema

- En el estado CONNECTED, que se produce tras el JOIN, hacemos un primer envío según el estado del sensor, que almacenamos negado en la variable puertaAbierta (recordemos que el sensor es NC está unido a la pull-down interna; en el pin leeremos HIGH cuando la puerta esté cerrada, y LOW cuando esté cerrada).
- Aprovechamos también para establecer el SF8.

```
228 switch (event) {
229     case CONNECTED:
230         printf("\r\n Connection - Successful \r\n");
231         /*
232         if (MBED_CONF_LORA_DUTY_CYCLE_ON) {
233             send_message();
234         } else {
235             ev_queue.call_every(TX_TIMER, send_message);
236         }
237         */
238         lorawan.set_datarate(DR_4); //SF8
239         puertaAbierta=!reed.read();
240         if(puertaAbierta){
241             enviarPuertaAbierta();
242         }else{
243             enviarPuertaCerrada();
244         }
245
246         break;
```



# Las funciones de envío de los uplinks

```
225 ✓ void enviarPuertaAbierta(){  
226     //Deshabilito la interrupción  
227     reed.fall(NULL);  
228     puertaAbierta=true;  
229     ev_queue.call(&send_message,1);  
230  
231 }  
232 ✓ void enviarPuertaCerrada(){  
233     //Deshabilito la interrupción  
234     reed.rise(NULL);  
235     puertaAbierta=false;  
236     ev_queue.call(&send_message,0);  
237 }  
238 ✓ void enviarHeartbeat(){  
239     ev_queue.call(&send_message,2);  
240 }  
241 ✓ /**  
242     * Event handler  
243     */
```

# La función send\_message()

```
163 static void send_message(int sensor_value){
164     //sensor_value puede ser 0 o 1, para indicar el estado de la puerta
165     //o 2 para indicar un uplink automático en respuesta a un downlink con fpending
166
167     DigitalOut LED(PA_8);
168     LED.write(1);
169     int16_t retcode;
170     lpp.reset();
171     if(sensor_value<2){
172         lpp.addDigitalInput(2, sensor_value);
173     }else{
174         //Si el mensaje no es de apertura/cierre enviamos el estado actual de la puerta
175         lpp.addDigitalInput(2, !reed.read());
176     }
177     LED.write(0);
178     retcode = lorawan.send(MBED_CONF_LORA_APP_PORT, lpp.getBuffer(), lpp.getSize(),
179                           uplinkACK==0?0x01:0x02);
180 }
```

```
181     if (retcode < 0) {
182         retcode == LORAWAN_STATUS_WOULD_BLOCK ? printf("send - WOULD BLOCK\r\n")
183         : printf("\r\n send() - Error code %d \r\n", retcode);
184
185         if (retcode == LORAWAN_STATUS_WOULD_BLOCK) {
186             //retry in 3 seconds
187             if (MBED_CONF_LORA_DUTY_CYCLE_ON) {
188                 //ev_queue.call_in(3s, send_message);
189             }
190         }
191         return;
192     }
193
194     printf("\r\n %d bytes scheduled for transmission \r\n", retcode);
195 }
```

# El estado TX\_DONE

```
264  ✓ case TX_DONE:
265      printf("\r\n Message Sent to Network Server \r\n");
266      //Después de cada envío reprogramamos el heartbeat
267      heartbeat.attach(&enviarHeartbeat, periodoHeartbeat*60.0);
268      ✓ if(puertaAbierta){
269          ✓ if(reed==1){
270              //La puerta se ha cerrado mientras enviabamos el mensaje de apertura
271              enviarPuertaCerrada();
272          }else{
273              //reed=0
274              //La puerta sigue abierta tras enviar el mensaje de apertura
275              reed.rise(&enviarPuertaCerrada);
276          }
277      } else{
278          ✓ if(reed==1){
279              //La puerta sigue cerrada tras enviar el mensaje de cierre
280              reed.fall(&enviarPuertaAbierta);
281          }else{
282              //La puerta se ha abierto mientras enviabamos el mensaje de cierre
283              enviarPuertaAbierta();
284          }
285      }
286      break;
287  case TX_TIMEOUT:
288  case TX_ERROR:
```

# Reenviar heartbeat en caso de error o uplink\_required

```
289 case TX_CRYPTO_ERROR:
290 case TX_SCHEDULING_ERROR:
291     printf("\r\n Transmission Error - EventCode = %d \r\n", event);
292     // try again
293     if (MBED_CONF_LORA_DUTY_CYCLE_ON) {
294         send_message(2);
295     }
296     break;
297 case RX_DONE:
298     printf("\r\n Received message from Network Server \r\n");
299     receive_message();
300     break;
301 case RX_TIMEOUT:
302 case RX_ERROR:
303     printf("\r\n Error in reception - Code = %d \r\n", event);
304     break;
305 case JOIN_FAILURE:
306     printf("\r\n OTAA Failed - Check Keys \r\n");
307     break;
308 case UPLINK_REQUIRED:
309     printf("\r\n Uplink required by NS \r\n");
310     if (MBED_CONF_LORA_DUTY_CYCLE_ON) {
311         send_message(2);
312     }
```

# RAK3172 – Reducir el consumo de emisión (1/2)

- Este modelo se comercializa optimizado para emitir en alta potencia (22dBm), pero en Europa sólo se permiten 14dBm.
- No tiene conectada la parte de baja potencia (LP), por lo que desafortunadamente no podemos acceder a un consumo reducido de 23.5mA@3.3V, y tenemos que conformarnos con 45.5mA@3.3V.
- Para acceder a este consumo intermedio tenemos que introducir una modificación en el archivo `mbed-os/connectivity/drivers/lora/TARGET_STM32WL/STM32WL_LoRaRadio.cpp`
- <https://forum.rakwireless.com/t/rak3172-too-much-consumption-in-transmit-eu868/4781>

Agosto 2021

Ya no hace falta hacer este cambio porque lo han introducido en el código oficial

# RAK3172 – Reducir el consumo de emisión (2/2)

Table 2. RF PA optimal settings

Mode	Output power (dBm)	Set_PaConfig()				SetTxParameters value (dBm)
		paDutyCycle	hpMax	deviceSel	paLut	
Low power (RFO_LP)	+15	0x06	0x00	0x01		+14
	+14	0x04				+14
	+10	0x01				+13
High power (RFO_HP)	+22	0x04	0x07	0x00	0x01	+22
	+20	0x03	0x05			
	+17	0x02	0x03			
	+14		0x02			

Agosto 2021  
Ya no hace falta hacer este cambio porque lo han introducido en el código oficial

```
STM32WL_LoRaRadio.cpp x mbed_app.json x LoRaWANInterface.cpp x
} else { // rfo_hp
    // Better resistance of the radio Tx to Antenna Mismatch
    // RegTxClampConfig = @address 0x08D8
    write_to_register(REG_TX_CLAMP, read_register(REG_TX_CLAMP) | (0x0F << 1));

    // if in mbed_app.json we have configured rf_switch_config in rfo_hp ONLY
    // so "stm32wl-lora-driver.rf_switch_config": "RBI_CONF_RFO_HP"
    // in this particular case it's not optimal settings for power<=20dBm
    if (board_rf_switch_config == RBI_CONF_RFO_HP) {
        // See Section 5.1.2 of the following Application Note
        // https://www.st.com/resource/en/application_note/an5457-rf-matching-n
        if (power > 20) {
            set_pa_config(0x04, 0x07, 0x00, 0x01);
        } else if (power > 17) {
            set_pa_config(0x03, 0x05, 0x00, 0x01);
        } else if (power > 14) {
            set_pa_config(0x02, 0x03, 0x00, 0x01);
        } else {
            set_pa_config(0x02, 0x02, 0x00, 0x01);
        }
    } else {
        set_pa_config(0x04, 0x07, 0x00, 0x01);
    }
}
```

# RAK3172 – Reducir el consumo de los dispositivos I2C en MbedOS (1/2)

- De los 3 puertos que incluye el RAK3172, el único capaz de entrar en modo de sueño (Deep sleep) es el I2C\_3
- Para que todos los puertos puedan entrar en Deep sleep tenemos que introducir una modificación en el archivo `mbed-os/targets/TARGET_STM/i2c_api.c`
- <https://github.com/ARMmbed/mbed-os/issues/15191>

# RAK3172 – Reducir el consumo de los dispositivos I2C en MbedOS (2/2)

```
545 void i2c_deinit_internal(i2c_t *obj)
546 {
547     struct i2c_s *obj_s = I2C_S(obj);
548
549     i2c_hw_reset(obj);
550
551     HAL_I2C_DeInit(&(obj_s->handle));
552
553     #if defined I2C1_BASE
554         if (obj_s->i2c == I2C_1) {
555             __HAL_RCC_I2C1_CLK_DISABLE();
556             #if defined(TARGET_STM32WL) || defined(TARGET_STM32WB)
557                 sleep_manager_unlock_deep_sleep();
558             #endif
559         }
560     #endif
561     #if defined I2C2_BASE
562         if (obj_s->i2c == I2C_2) {
563             __HAL_RCC_I2C2_CLK_DISABLE();
564             #if defined(TARGET_STM32WL)
565                 sleep_manager_unlock_deep_sleep();
566             #endif
567         }
568     #endif
569 }
```



# RAK3172 – Usar la Flash a modo de EEPROM

Usamos la librería kvstore, que se basa en pares clave/valor

```
207 //Obtenego la configuración (periodo heartbeat (hb) y uplink con (ack))
208 int res = MBED_ERROR_NOT_READY;
209 char kv_key[] = {"/kv/hbm"};
210 char kv_value[5];
211 char str[5];
212 kv_info_t info;
213
214 kv_get_info(kv_key, &info);
215 //printf("kv_get_info key: %s\n", kv_key);
216 //printf("kv_get_info info - size: %u, flags: %lu\n", info.size, info.flags);
217
218 res = kv_get(kv_key, kv_value, info.size, 0);
219 if(res == MBED_SUCCESS){
220     printf("Periodo heartbeat: [%s] minutos\n", kv_value);
221     sscanf(kv_value, "%d", &periodoHeartbeat);
222 }else{
223     strncpy(str, "30", 5);
224     kv_set(kv_key, str, strlen(str), 0);
225     printf("Estableciendo periodo heartbeat predeterminado: 30 minutos\n");
226     periodoHeartbeat=30;
227 }
```

# RAK3172 – Medir la tensión de alimentación

Usamos la referencia interna

```
123 //Calculate Voltage data
124
125 //Vdd:      CPU Vdd line voltage
126 //          (If VREF+ connected Vdd and VREF- connected GND)
127 //dt_adc:   ADC data (12bit)
128 //VREFINT:  Internal bandgap voltage
129 //          1.182V(min), 1.212V(typ), 1.232V(max)
130 //VREFINT_CAL Raw data acquired at temperature of 30 °C,
131 //          Vdd = 3.3 V Addr: 0x1FFF75AA-0x1FFF75AB
132
133 //VREFINT_CAL = 3000 * VREF(Factory) / 4096      [V]
134 //Vdd         = VREFINT_CAL / VREFINT(ADC float) [V]
135 AnalogIn bat(ADC_VREF);
136 uint16_t     vref_cal;
137 double       vdd_calibed;
138 double       vref_calibed;
139 vref_cal = *(__IO uint16_t *)((uint32_t)0x1fff75aa);
140 vref_calibed = 3.3f * (double)vref_cal / 4096.0f;
141
142 vdd_calibed = vref_calibed / bat.read();
```