# Uso de RIOT-OS para programación en el ámbito de IoT

María Ariza Gamero

Alejandro Hernán Luque

Alberto Vázquez Baeza

Infraestructura Avanzada de Redes de Sensores

Máster en Sistemas Inteligentes en Energía y Transporte

# Contenido

# 1. Presentación

Se desea instalar el sistema operativo RIOT-OS en las placas de Texas Instruments cc2538dk con el programador SmartRF06 y cc2650stk con el programador xds110, conocida como Sensor Tag. Para ello, será necesito el uso de Docker. Una vez ejecutado RIOT-OS, se procederá a comprobar que funciona correctamente empleando un sencillo programa de encendido de LEDs, que primero se compilará desde RIOT-OS y después de flasheará una herramienta nativa propietaria y necesaria de Texas Instrument, Uniflash. Otro flasheador propio de esta misma marca es Flash Programmer, que también se usará. Por último, se procederá a realizar comunicación usando el protocolo 802.15.4.

Docker está disponible para Windows y para Linux, pero en Windows necesita la versión Pro, Enterprise o Education para que tenga capacidad de virtualización de forma nativa. Dado que todos los dispositivos no se poseen esta versión, se ha decidido emplear Linux en su versión Ubuntu 18.04.2 LTS 64-bit desde nativo y en máquina virtual, la versión Ubuntu 16.04.5 LTS 64-bits, además de llegar a compilar un programa de test de prueba de leds en el microcontrolador cc2538dk en Windows 10.

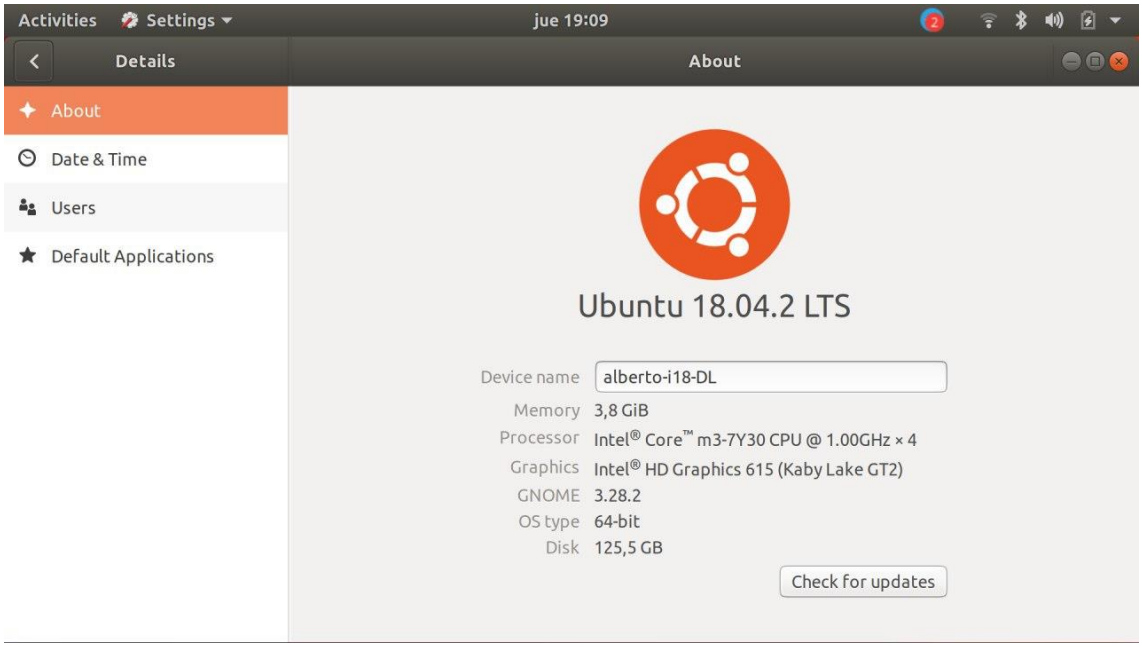


*Figura 1. Detalles Ubuntu 18.04.2 en nativo.*

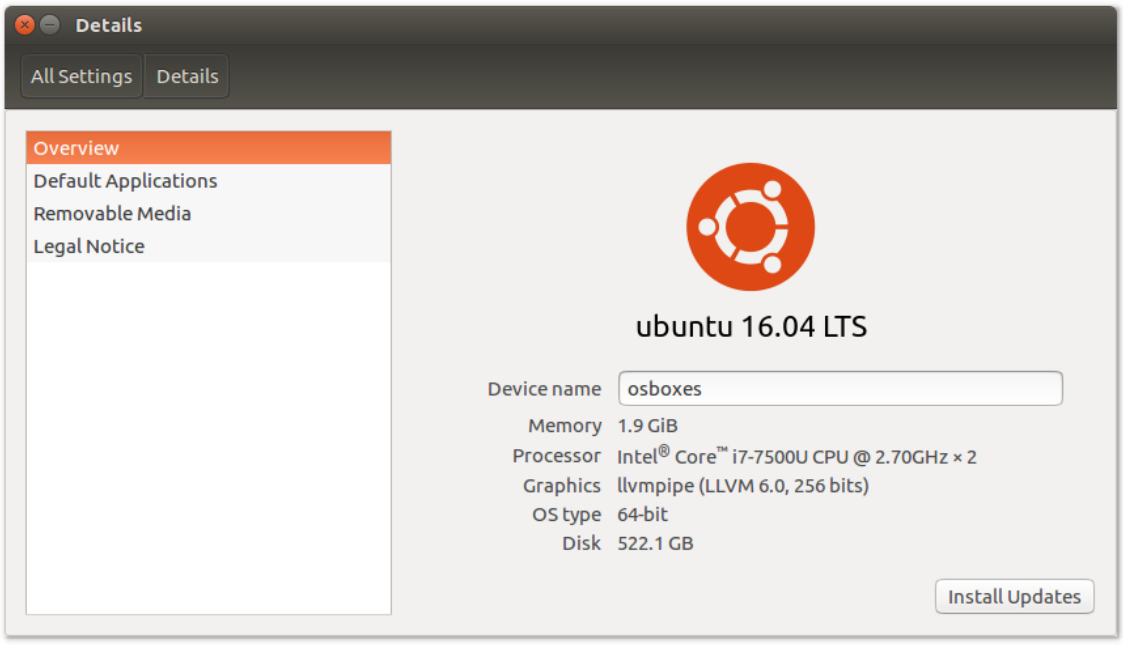


*Figura 2.. Detalles Ubuntu 16.04.5 en máquina virtual.*

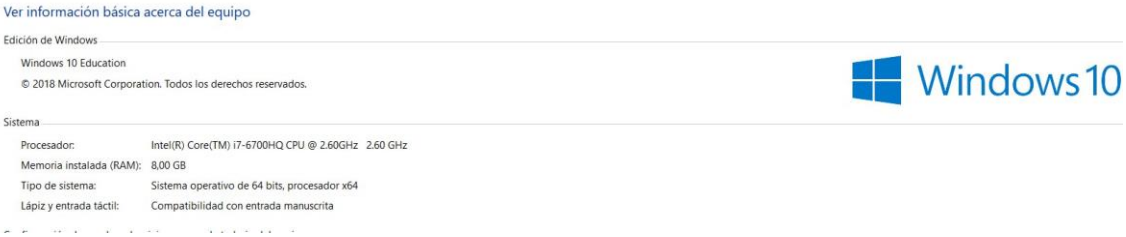


*Figura 3. Detalles Windows 10 Education en nativo.*

## 2. Git

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la fiabilidad del control de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos en ordenadores y coordinar el trabajo que varias personas realizan sobre los archivos compartidos. Git modela sus datos como un conjunto de instantáneas de un sistema de archivos. Cada vez que se confirma un cambio o se guarda el estado del proyecto en Git, se hace una instantánea del aspecto de los archivos en ese momento y se guardar la referencia a dicha instantánea. Git solo almacena los archivos que se han modificado, creando un enlace al archivo anterior si el nuevo subido es idéntico. De esta forma se consigue aumentar la eficiencia y se reduce el consumo de espacio. Otra característica destacable de Git es su capacidad para operar de forma local, no necesita información de ningún otro ordenador de la red. Al tener la información en el disco local, la mayoría de las operaciones son prácticamente inmediatas.

Se usará en este proyecto esta herramienta para el control de versiones y compartir archivos, de forma que todas las personas del proyecto puedan trabajar de forma simultánea. Una vez

instalado Git en el dispositivo local, se crea la carpeta donde se quiere crear el repositorio. Desde dicha carpeta, donde estará contenido el proyecto, se ejecuta el *Git bash*.

## 2.1. Conectarse a este repositorio

### Iniciar el repositorio de Git

El `git init` se realiza desde el terminal si se encuentra en Linux o en el bash de Git si se encuentra en Windows.

```
>> git init
```

Una vez se ejecuta este comando se crea una carpeta .git en la carpeta de trabajo.

### Conectarse al repositorio

Una vez iniciado el Git, se ejecuta la siguiente sentencia para conectarse a este repositorio. Al final de la url tiene que aparecer siempre .git.

```
>> git remote add origin https://github.com/blalebla/IARS.git
```

Se suele utilizar `origin` como etiqueta de carpeta de origen. Se ha conectado con el repositorio pero no se ha descargado nada todavía.

A continuación, se ejecuta el siguiente comando:

```
>> git fetch origin
```

Fetch hace una comprobación del repositorio online, enumero los objetos y las ramas presenters, en este caso solo se tiene la rama master.

### Descarga archivos del repositorio

Para descargar los archivos del repositorio se realiza un `pull:`

```
>> git pull origin master
```

Se ha realizado un `pull` de la rama master del repositorio online.

Una vez realizado lo anterior ya se tiene acceso de forma local a los archivos.

## 2.2. Realizar cambios en Git

Una vez se ha descargado el repositorio en el que se desea trabajar, ya se pueden realizar modificaciones, crear archivos nuevos, eliminar algunos exitentes, crear carpetas...

### Estructura de Git

Las versiones en Git se distribuyen en 3 espacios:

- **Working Directory**: es el directorio donde se trabaja, todo lo que hay aquí se puede perder.
- **Staging**: es la zona de preparado
- **Commit**: la zona donde se guardan directamente

### Pasar de WD al área de Staging

Para pasar del working directory al área de Staging se hace uso del siguiente comando:

```
>> gitt add .
```

El punto lo que hace es agregar al área de staging todos los archivos que se han creado o modificado. También se pueden añadir los archivos uno a uno, o declarándolos de forma consecutiva tras el *add*.

```
>> git add <archivo1.md> <archivo2.bin>
```

#### Commit

Para realizar un *commit* se realiza la siguiente operación:

```
>> git commit -m "comentario del comit"
```

### 2.3. Realizar subida de los cambios realizados

Para subir los cambios realizados se debe realizar un *push* de la siguiente manera:

```
>> git push origin master
```

Una vez ejecutado este comando suele pedir las credenciales de Github.

### 2.4. Clonar repositorio de RIOT

Para clonar un repositorio desde el sitio web de github, es necesario estar desde dentro de la carpeta donde se desea tener clonado este repositorio. Esta carpeta podrá clonar desde Linux o desde RIOT mediante la sentencia `git clone` seguido de la dirección web que contiene el repositorio:

```
>> git clone https://github.com/RIOT-OS/RIOT.git
```

Con esta sentencia se clonará la carpeta completa con el contenido. Si se añade al final de la sentencia un espacio y '.', se clonará el contenido de la carpeta sin la carpeta predecesora que lo contiene.

```
>> git clone https://github.com/RIOT-OS/RIOT.git .
```

Si la carpeta se clona desde Linux, se tendrá permiso de administrador y se podrán hacer modificaciones sobre esta. Si por el contrario se hace desde RIOT, hay que hacerlo desde la carpeta que se ha enlazado con el host al crear el container. En este caso, la carpeta aparecerá bloqueada desde el host.

## 3. Hardware

Para el desarrollo de este trabajo se ha contado con el siguiente hardware:

### 3.1. cc2538dk

Es la misma placa que se ha usado en las prácticas de la asignatura y cuenta con un módulo de expansion (evaluation board SRF06). El microprocesador es un cortex M3 de ARM y cuenta con

display, botonera de 5 botones, 4 leds de distintos colores, sensor temperatura, de luz, comunicación 802.15.4..... Esta placa cuenta con programador integrado (un xds1100 v3).


*Figura 4. Microcontrolador cc2538dk con SmartRF06.*

## 3.2. cc2650stk: Sensor Tag

Este módulo pensado para IoT de Texas Instrument cuenta con el micro cc2650stk, que está basado en la arquitectura ARM con un cortex M3 y distintos sensores (luz, temperatura, acelerómetro, micrófono...). Como GPIOs que interactúan con el usuario, tiene dos botones y dos ledes, uno verde y uno rojo. También dispone comunicación bluetooth low energy y 802.15.4. Para la alimentación del Sensor Tag se necesita una pila cr2032 (comúnmente llamadas de botón). Para programarla es necesario un programador externo, el xds1100 v3.

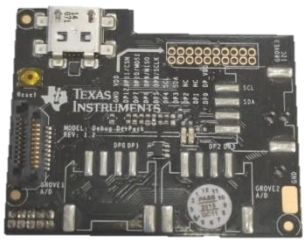

*Figura 5. Microcontrolador cc2650stk (Sensor Tag).*

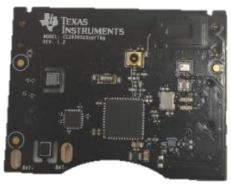

*Figura 6. Programador xds110 para Sensor Tag.*

### 3.3. cc2531: Sniffer

Para monitorizar las comunicaciones inalámbricas se cuenta con un sniffer, también de TI. Gracias a este, se puede comprobar que las comunicaciones entre ambas placas son correctas.

*Figura 7. Sniffer cc2531.*

## 4. Docker

Docker es un proyecto open source para automatizar la implementación de aplicaciones como contenedores portátiles y autosuficientes que se pueden ejecutar en la nube o localmente.

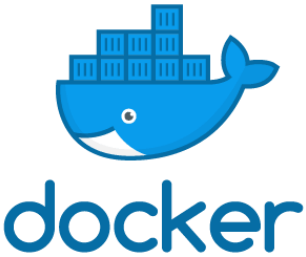


*Figura 8. Logo de Docker.*

Existes tres conceptos que hay que tener claros a la hora de trabajar con Docker. Una **imagen de contenedor Docker** es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones. Un **contenedor** es una unidad estándar de software que empaqueta tanto el código y como todas sus dependencias para que la aplicación se ejecute de un entorno a otro de forma rápida y fiable. Para compartir archivos entre el container y el host, es necesario crear un volumen. El **volumen** será el enlace que comunique la carpeta en host y la carpeta en Docker, de modo que este será borrado una vez se salga del contenedor de Docker, permaneciendo los archivos compartidos.

El software en contenedores siempre se ejecutará de la misma manera, independientemente de la infraestructura. Los contenedores aíslan el software de su entorno y garantizan que funcione de manera uniforme a pesar de las diferencias, por ejemplo, entre el desarrollo y la organización. Funcionan tanto en Windows como en Linux.

Los contenedores Docker se caracterizan principalmente porque son:

- **Estándares:** Docker creó el estándar de la industria para contenedores, por lo que podrían ser portátiles en cualquier lugar
- **Ligeros:** los contenedores comparten el kernel del sistema operativo de la máquina y, por lo tanto, no requieren un sistema operativo por aplicación, lo que aumenta la eficiencia del servidor y reduce los costos del servidor y de la licencia.

- **Seguros:** las aplicaciones son más seguras en contenedores y Docker proporciona las capacidades de aislamiento predeterminadas más sólidas de la industria.

Los contenedores de Docker surgen a partir de la tecnología LXC de Linux. Docker, además de ser un sistema de virtualización ligera para ejecutar contenedores, ofrece además la capacidad para crear y diseñar contenedores o enviar imágenes y crear versiones de estas entre otras muchas funciones.

Intentar encontrar similitudes entre Docker y una máquina virtual es un error, pues son dos tecnologías diferentes que surgen a raíz de intentar resolver diferentes problemas.

Por un lado, las máquinas virtuales nacen para plantear solución a la utilización en los servidores y fundamentalmente abstraer el hardware del sistema operativo, creando un entorno aislado para los sistemas operativos y aplicaciones. De esta manera es posible tener en un único servidor físico varias máquinas virtuales. Por otro lado, los contenedores abstraen aplicaciones del sistema operativo para hacer las aplicaciones más portables. Los contenedores Docker arrancan más rápido, ocupan menos espacio en el disco y permiten ejecutar más contenedores Docker que máquinas virtuales en un servidor. Con las máquinas virtuales pueden tenerse varios sistemas operativos en un mismo servidor, existiendo además numerosas plataformas de virtualización. Dependerá del problema que se plantea el usar una solución u otra, pues son soluciones diferentes a problemas diferentes.

## 4.1. Instalar Docker

Existen dos métodos para instalar Docker en Ubuntu 16.04. Un método consiste en instalarlo en una instalación existente del sistema operativo. El otro implica correr un servidor con una herramienta llamada Docker Machine que instala automáticamente Docker en él.

En esta aplicación se ha optado por instalarlo y utilizarlo a través de Ubuntu. Basta con emplear el siguiente comando desde el terminal de Linux para tener instalado Docker:

```
>> sudo upt install docker.io
```

Para instalarlo desde Windows, basta con instalar la aplicación de forma similar a cualquier otro programa en este sistema operativo.

## 4.2. Crear un container

Para crear un container, se emplea el comando docker run.

```
>> sudo docker run
```

Para que el container ejecute una imagen, debe primero obtenerse la imagen y posteriormente ejecutarla incluyendo el nombre de la imagen a continuación del comando run, como se explica en el apartado 5. RIOT-OS con la imagen de este sistema operativo.

# 5. RIOT-OS

RIOT alimenta el Internet de las Cosas como Linux alimenta el Internet. RIOT-OS es un sistema operativo open source y gratuito desarrollado por una comunidad que reúne compañías, instituciones académicas y aficionados distribuidos por todo el mundo.


*Figura 9. Logo de RIOT-OS.*

RIOT es compatible con la mayoría de los dispositivos IoT de bajo consumo y las arquitecturas de microcontroladores. RIOT tiene como objetivo implementar todos los estándares abiertos relevantes que respaldan un Internet de las Cosas conectado, seguro, duradero y amigable con la privacidad.

La semilla de RIOT fue FeuerWare, un sistema operativo que nace en 2008 para redes de sensores inalámbricos. Surge como parte del proyecto de FeuerWare donde se buscaba la monitorización de los bomberos. Para aumentar la modularidad e incluir nuevos protocolos IETF, desde el repositorio original de FeuerWare se bifurcó µkleos. El soporte para 6LoWPAN, RPL y TCP se integró en los años siguientes. En 2013 RIOT se hace público siendo el sucesor directo de µkleos, promoviendo RIOT a una comunidad mucho más grande.

Actualmente, RIOT se ejecuta en varias plataformas, incluidos dispositivos integrados y PC comunes. El código dependiente del hardware se reduce al mínimo y se abstrae del propio kernel. Esta solución diseñada pensando en IoT posee una arquitectura de microkernel y emplea estándares como 6LoWPAN que facilita su implementación.

## 5.1. Descargar la imagen de RIOT-OS

Para poder ejecutar la imagen de RIOT, primero hay que descargarla a través del comando pull:

```
>> sudo docker pull riot/riotbuild
```

## 5.2. Ejecutar la imagen de RIOT-OS

Una vez se ha descargado la imagen, hay que crear un container, de forma similar a como se ha explicado en el apartado 4.2. Crear un container, y ejecutar la imagen indicando su nombre tras ejecutar el contenedor. Con el comando -ti se abre el terminal de RIOT-OS tras ejecutar la imagen.

```
>> sudo docker run -ti riot/riotbuild
```

## 5.3. Crear un volumen compartido entre RIOT y el host

Al ejecutar el container con la imagen de RIOT no se tiene acceso a los archivos creados dentro del container. Cuando se sale del container se pierde la imagen con todos los archivos nuevos o modificados. Por lo que es necesario crear un volumen que comparta estos archivos con el host para una vez fuera del container no perderlos. Para crear el volumen es necesario estar corriendo en Docker la imagen de RIOT. La sentencia empleada es la siguiente:

```
>> sudo docker run -ti –rm –name RIOT1 -v $PWD/IARS:/doc riot/riotbuild
```

Con el comando `–rm` (remove) se elimina el volumen una vez se ha salido de RIOT. Seguidamente, se le da nombre al volumen, en este ejemplo RIOT1. Con el comando `-v` se crea el volumen. A continuación, se indica el nombre de la carpeta compartida en el host seguido ':/' y el nombre de la carpeta compartida en RIOT. Cuando se sale de RIOT se elimina el volumen pero no los archivos compartidos.

# 6. Compilar

Una vez está listo el sistema operativo RIOT-OS, se quiere comprobar si funciona haciendo compilar un programa en nativo que imprima por pantalla "Hello, World!". Posteriormente, se compilará un programa de test que incluye el repositorio de RIOT bajado de github. Este programa hará una lectura de los leds y los encenderá y apagará simultáneamente.

Para compilar es necesario tener dentro de la misma carpeta que contiene el archivo con el código que se compilará el archivo makefiel. El archivo makefile necesita definir las siguientes macros:

- Application: debe contener el nombre de la aplicación.
- Riotbase: especifica la ruta en el repositorio de RIOT.
- Board: se inicializa en nativo por defecto.

Si se utilizan funcionen específicas que no son genéricas de todas las placas, es necesario definirlas dentro de este archivo con la función usemodule.

## 6.1. Compilar en nativo

Con el comando `make` se compila el archivo main.c con el código de la función. Para ello es necesario ejecutar dicho comando desde dentro de la carpeta que contiene tal archivo.

```
>> sudo make BUILD_IN_DOCKER=1 DOCKER=”sudo docker” term
```

## 6.2. Compilar el microcontrolador

Para compilar el programa en el microcontrolador, basta con incluir el comando `BOARD` seguido del nombre de la placa.

```
>> sudo BOARD=cc2650stk make BUILD_IN_DOCKER=1 DOCKER=”sudo docker”
```

```
>> sudo BOARD=cc2538dk make BUILD_IN_DOCKER=1 DOCKER=”sudo docker”
```

Si se añade el comando `term` dentro de la sentencia, se abre un terminal al compilar para ver la comunicación del terminal con la placa. En nativo es posible, pero en este caso no es posible al no tener disposición de puerto serie.

# 7. Flashear

Para poder flashear los microcontroladores, esto es, almacenar en la memoria flash el programa que previamente se ha compilado, pueden emplearse dos métodos diferentes: desde el propio terminal con el comando `make flash` o desde una de las dos herramientas de Texas Instrument, Uniflash o Flash Programmer.

## 7.1. make flash

En un primer momento se plantea el uso del comando *make flash* propio de RIOT para poder compilar.

```
>> make flash BOARD=cc2538dk
```

```
>> make flash BOARD=cc2650stk
```

Al intentar flashear cualquiera de las dos placas, se descubre que los microcontroladores de Texas Instruments precisan de su propio flasheador. El primer flasheador, Uniflash, está disponible tanto para Windows como para Linux, y el segundo, Flash Programmer solo para Windows.

## 7.2. Uniflash

Aunque directamente se puede flashear desde el host, la idea es hacer que RIOT tenga acceso a Uniflash para poder flashear desde RIOT. Para que RIOT tenga acceso a Uniflash en un primer momento se crea un volumen que comparta esta herramienta. Posteriormente se descubre que el comando coge directamente, sin necesidad de crear el volumen, la aplicación de Uniflash pasándole la ubicación del archivo.

*Figura 10. Logo de Uniflash.*

Es necesario estar dentro de la carpeta que contiene el archivo main.c que se quiere compilar y flashear.

Para compilar desde RIOT-OS a la vez que se flashea, se emplea la siguiente sentencia:

```
>> sudo BOARD=cc2650stk make BUILD_IN_DOCKER=1 DOCKER=”sudo docker” UNIFLASH_PATH=/home/osboxes/ti/uniflash_5.0.0/ flash
```

Los comandos contenidos entre el nombre de la placa (*BOARD=cc2650stk*) y *flash* pueden cambiarse de orden, pues son comandos independientes.

Esta sentencia solo funciona para la Sensor Tag y no para la cc2538dk, que necesita ser flasheada desde el propio programa Uniflash. Funciona tanto desde Ubuntu en nativo como en máquina virtual.

Para ambas placas, puede usarse el programa Uniflash en nativo desde Ubuntu o Windows, aunque en este último caso da error de manera ocasional. El programa detecta el microcontrolador de forma automática cuando se conecta. Una vez se ha compilado el archivo

main.c y se han obtenido los archivos .bin, .elf y .hex, debe introducirse uno de estos tres para flashear la placa.

Puesto que Uniflash no funciona desde máquina virtual, se instala en Windows, que en ocasiones también falla, por lo que se recurre a otro flasheador también de Texas Instrument: Flash Programmer.

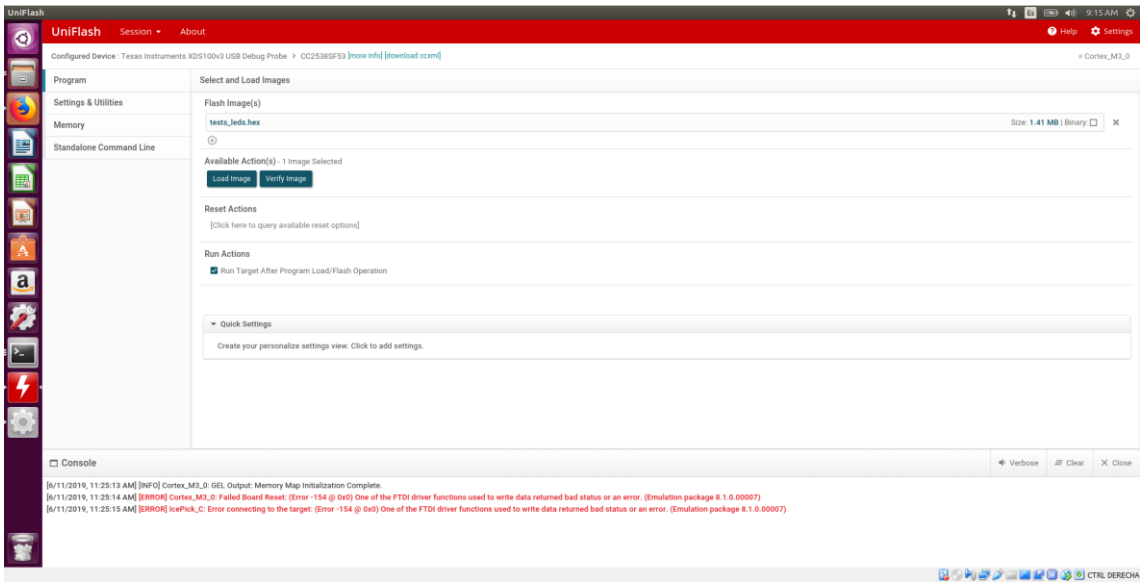


*Figura 11. Uniflash en Ubuntu virtual.*

## 7.3. Flash Programmer

Flash Programmer está disponible solo para Windows y funciona de forma similar a Uniflash. El mismo programa detecta de forma automática la placa cuando se conecta, se introduce uno de los tres archivos compilados y solo hay que dar a *play* para flashear el microcontrolador.

Hay que tener en cuenta que si el path del archivo para flashear es demasiado largo puede da error, por lo que la forma más fácil y rápida de ejecutarlo es contener dichos archivos dentro de la unidad C.

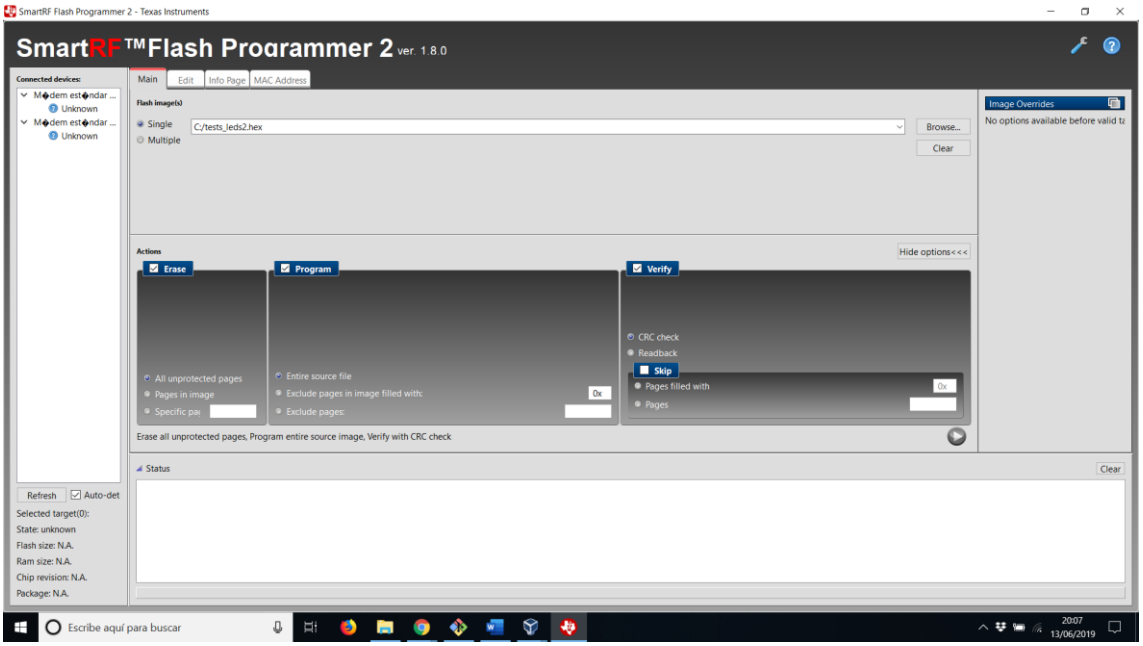


*Figura 12. Flash Programmer en Windows nativo.*

## 8. Programa leds

Para realizar un "Hello world" y comprobar que se tiene un control básico e inicial sobre las placas que se van a usar, se parte del código de test (incluido en el repositorio de RIOT) de los leds en la carpeta test/leds.

Con ese ejemplo como base, se programa un toggle de los leds 0 y 1 para asegurar que los conocimientos necesarios han sido adquiridos, así, del archivo original solo se toma la función dumb_delay, que realiza la espera necesaria para que el ojo humano pueda ver los cambios en los leds.

La programación resulta tan sencilla como escribir el comando toggle del led, sin realizar una inicialización o configuración de las GPIO, que ya son realizadas en los archivos pertinentes de cada board y que se utilizan al compilar para cada placa. De esta forma, el mismo ejemplo que se ha escrito para realizar el "Hello world" en la sensor tag (cc2650stk) se puede usar en la placa de prácticas (cc2538dk con evaluation board).

## 9. Botones

Para realizar el programa de control de botones, se empieza en un paso previo, que es comprobar que se inicializan bien. Para ello, se parte del ejemplo de test/buttons, en el que se comprueban uno a uno los botones y se manda un mensaje por el puerto serie en el que se dice cuántos se han encontrado. Se modifica para que cuando se encuentra un botón, se encienda

un led. El programa comprueba si se puede configurar una interrupción para saber si ese botón existe en la placa.

Al compilar y cargar el programa en la cc2650stk, el código se ejecuta correctamente, encendiendo los dos leds de los que dispone la placa, para informar de que tiene dos botones.

Al tratar de compilar el mismo código para la cc2538dk, ocurre que los botones no están definidos y da error, así que toca navegar por los archivos de configuración de la placa (en /boards y /cpu) para comprobar qué ocurre.

Se detecta que, para esta placa no se han definido ni inicializado los botones en ningún sitio, así que se procede a definir en qué puerto y qué pines corresponden a cada botón y para ello se usa la documentación oficial.

Con todos los botones debidamente configurados, se corre el ejemplo y se comprueba que todo está en orden antes de dar el siguiente paso, control de los leds mediante los botones.

Para ello, se configura una interrupción hardware cuando se detecta un flanco de bajada en un botón, que llama a una función de callback que realiza el toggle del led correspondiente. Este ejemplo se compila y flashea para dos botones y ledes en la cc2650stk y una vez se comprueba su correcto funcionamiento, se modifica el código anterior para que funcione con 5 botones en lugar de con solo dos (se hacen dos main.c diferenciados, ya que con el original se quedarían 3 botones sin programar y con uno completo, daría error por falta de botones en la sensortag). El código específico de la cc2538dk, asigna la capacidad de toggle de cada led a un botón y el quinto botón, hace toggle a todos a la vez.

Una vez terminados estos ejemplos, se tiene un control sobre RIOT y sobre ambas placas suficiente como para dar el siguiente paso.

## 10. Comunicación

Tanto la cc2538dk con evaluation board como la cc2650stk (SensorTag), tienen capacidades de comunicación a través de 802.15.4 (la SensorTag además, puede comunicarse por BLE). El objetivo final sería crear una comunicación estable entre ambas placas, de forma que se pueda controlar una desde la otra.

Para ello, habría que inicializar el módulo de radio presente en las dos placas de TI y establecer una serie de interrupciones que, al recibir un determinado comando por radio, se ejecute una acción determinada. El ejemplo sería que el control de los leds de una placa recaiga en la botonera de la otra y viceversa. De esta forma se tendría creada una red IoT inalámbrica básica.

El primer paso, tras realizar toda la inicialización y configuración sería lanzar mensajes y comprobar que se están enviando correctamente. Para ello se hace uso de un sniffer cc2531 de TI y wireshark para escuchar todo lo que se emite a través del 802.15.4, tal y como se hizo en las prácticas de las asignaturas. Este paso no se ha logrado durante las pruebas y no se ha llegado a escuchar nada, pese a que el sniffer estaba funcionando correctamente (se lograba ver tramas 802.15.4 cerca del pasillo del DTE). Pese a la búsqueda de solución y escrutinio del código usado

y de la documentación de TI, no se ha logrado hacer funcionar el módulo inalámbrico de ninguna de las dos placas.

Una vez ha funcionado el paso anterior, se tendría que haber establecido un sistema de comunicación entre ambas placas y definido un sistema de órdenes para que, al llegar un mensaje, se ejecutase una acción (encender un led) de forma que se tenga feedback de que todo funciona correctamente.

Con lo anterior en funcionamiento, el objetivo final se habría completado, ya que un mensaje que se envía al pulsar un botón y que al llegar a su destino desencadena una acción (encender un led), era el fin último del presente trabajo.

# 11. Finalización del proyecto y futuras mejoras

Al finalizar el proyecto, se han aprendido a usar Docker, RIOT-OS, los flasehadores de TI Uniflash y Flash Programmer, Wireshark y Git como software y como hardware los microcontroladores cc2538dk y cc2650stk y el sniffer cc2531.

El esquema final del Git del proyecto se muestra en la siguiente figura:

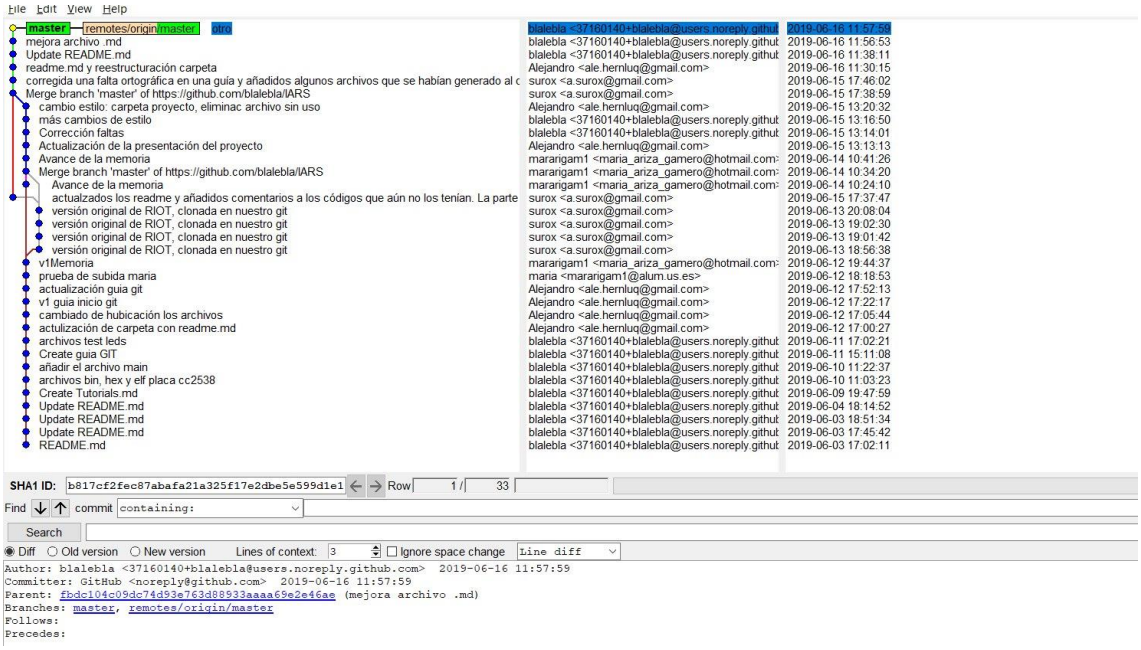


*Figura 13. Modificaciones de Git del proyecto.*

Con las comunicaciones entre placas ya establecidas, se pueden realizar multitud de proyectos. Por ejemplo y gracias al sensor de temperatura de la SensorTag, se podría automatizar el encendido de la caldera (led) en la cc2538dk, cuando la temperatura ambiente baje de un punto concreto, o el encendido de la luminaria en base al sensor de luz.