Universidad de Cantabria

Facultad
de
Ciencias

MARCO PARA EL DESARROLLO DE APLICACIONES C SOBRE PLACAS MICRO:BIT

(Framework for the development of C applications on Micro:bit boards)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Noé Ruano Gutiérrez

Director: Héctor Pérez Tijero

Co-Director: Mario Aldea Rivas

Julio - 2023

Resumen.

En 2015 la BBC lanzó la primera versión de Micro:bit, un computador de hardware libre, sencillo y de muy reducidas dimensiones, con el objetivo de acercar la programación a los niños y jóvenes de Reino Unido [1]. La sencillez de este tipo de hardware y, por ende, su bajo costo, lo hacen muy accesible, y el candidato perfecto para integrarlo en los métodos docentes de un amplio abanico de niveles de enseñanza. Tanto es así, que Micro:bit pasará a formar parte del conjunto de recursos con que se abordará la puesta en práctica, por parte del alumnado, de algunos de los conceptos de programación en lenguaje C impartidos en la asignatura de Introducción al Software, en el Grado en Ingeniería Informática.

El presente documento recoge la síntesis del proceso de desarrollo de una librería de alto nivel con la que los alumnos podrán interactuar de una manera sencilla con el hardware, así como el conjunto de herramientas con las que podrán construir, cargar y poner a prueba sus códigos en el SBC (*Single Board Computer*). Asimismo, se detallan la motivación primera del proyecto y la metodología y tecnologías empleadas en el desarrollo de este y de su documentación asociada, además de las pruebas realizadas sobre el hardware, y la manera en la que se ha procurado facilitar la distribución de toda la utilería software elaborada, gracias a la cual los alumnos podrán realizar sus propios desarrollos sobre la plataforma Micro:bit.

Palabras clave: Micro:bit, C, API, Librería, Software Embebido

Abstract.

In 2015 the BBC launched the first version of Micro:bit, a simple, compact and free-hardware-based computer, with the aim of giving kids and teens all around the UK their first approach to computer programming. Its simplicity and, therefore, its low cost, makes this kind of hardware quite accessible, and the perfect candidate to be integrated into the teaching methods of a wide range of educational levels. And that is the reason why Micro:bit will become part of the set of resources with which students will take a practical approach on some of the C programming language concepts that are taught in the subject Introduction to Software, which is part of the Bachelor's Degree in Computer Engineering.

This document sums up the development process of a high-level library with which students will be able to interact with the hardware in a simple manner, as well as the set of tools with which they will be able to build, load and test their codes on the SBC (*Single Board Computer*). Likewise, the project's main motivation is also pointed out, as well as the methodology and technologies used in its development and documentation. Lastly, it also contains the set of tests that were carried out on the hardware, and remarks the way in which it was made easy for students to acquire all the software tools that will serve them to make their own software developments for the Micro:bit platform.

Keywords: Micro:bit, C, API, Library, Embedded Software

Agradecimientos

Agradecer a mi familia su apoyo constante a lo largo de esta y todas las etapas de mi vida.

También agradecer a los directores del proyecto su implicación para con este.

Por último, agradecer a todos los buenos amigos que he tenido la gran suerte de hacer a lo largo de estos cuatro años, el tiempo y buenos momentos que hemos podido compartir juntos. Eternamente agradecido por todo el bien que me han hecho.

Índice

Índice de figuras

1. Introducción

1.1. Motivación

Micro:bit es una plataforma hardware de bajo coste desarrollada por la BBC en colaboración con empresas del sector tecnológico como Microsoft, Nordic o ARM entre otros, así como la Universidad de Lancaster, y que tiene como objetivo el acercamiento de los niños y jóvenes al mundo del desarrollo software, con las implicaciones positivas que ello tiene sobre facultades como son la creatividad o la lógica. No obstante, este hardware puede también resultar de utilidad en la docencia de asignaturas en niveles superiores de educación, como es el caso que nos ocupa.

Tradicionalmente, las asignaturas de programación siguen una dinámica basada en clases magistrales donde se imparten los conceptos inherentes al lenguaje sobre el que versen dichas clases, además de clases prácticas donde se refuerza la asimilación de dichos conceptos a través de la realización de ejercicios. No obstante, y a pesar de haber quedado más que demostrada la efectividad de este sistema a lo largo de los años, la introducción de nuevas metodologías en la docencia de la programación podría llegar a mejorarlo significativamente, al menos en lo que respecta a la forma en que los alumnos asimilan la información que les es expuesta. Y ese es el objetivo primordial de este proyecto, el proveer a los estudiantes del conjunto de herramientas con el que puedan aplicar sus conocimientos de programación sobre hardware real, en aras de que las sesiones prácticas les resulten mucho más motivadoras, y puedan conocer parte de las aplicaciones reales que sus conocimientos podrían llegar a tener. Concretamente, se pretende dotar de estas capacidades a los alumnos de la asignatura de Introducción al Software, en la cual se imparten aspectos básicos de la programación como tipos de datos, sentencias condicionales, bucles, etc.

A tal efecto, se ha desarrollado una librería de alto nivel con la que poder interactuar de una manera sencilla e intuitiva con el hardware presente en el Micro:bit, partiendo de la funcionalidad provista por un sistema operativo embebido (Microbian), desarrollado y mantenido por el profesor Mike Spivey, de la Universidad de Oxford [2]. Se trata de un sistema operativo embebido y multiproceso, el cual proporciona el conjunto de llamadas necesario para interactuar a bajo nivel con el hardware de la placa.

Fruto de la complejidad que un estudiante neófito en la programación pudiera encontrar en la interacción con el mencionado hardware, así como la complejidad inherente a la programación de sistemas empotrados, surge la necesidad de dotarle de un conjunto de llamadas que consigan abstraerlo de conocer los detalles sobre su funcionamiento, y le doten de las mismas o muy similares capacidades para hacer un uso lo más completo posible del abanico de dispositivos a su disposición. Se han desarrollado, además, herramientas auxiliares que les permitirán construir y cargar sus desarrollos en la placa, con objeto de suprimir en la medida de lo posible la complejidad de dichas operaciones.

1.2. Objetivos

El proyecto tiene como objetivo, primero, el desarrollo de una API de alto nivel (la cual denominaremos librería “Ubit”) con la que poder elaborar programas que interactúen con el hardware presente en el SBC a través del sistema operativo Microbian, proporcionando de este modo una abstracción de este y sus funciones, y reduciendo la complejidad en el manejo de los distintos dispositivos presentes en la placa. Por medio de esta API, se pretende facilitar la práctica de conceptos introductorios de programación, por lo que, en el diseño de la interfaz de

alto nivel, primará la facilidad de uso sobre otros aspectos como la eficiencia (p.ej., mediante el uso del paso por valor en lugar de punteros).

Además, se proporcionará junto con la librería utilería adicional que simplifique los procesos de compilación y carga de código, y oculte al alumno los aspectos relativos a estos procesos que pudieran resultar más avanzados, y los cuales no son vistos en la asignatura de Introducción al Software. No obstante, sí se requerirá introducir las opciones de compilación y enlazado que sí se ven en dicha asignatura.

Por último, se proporcionará la documentación asociada a todos y cada uno de los desarrollos software realizados, y se desarrollará el conjunto de programas de prueba con que poder verificar el correcto funcionamiento de cada una de las funciones de que se compone la librería, además de un prototipo de práctica para la asignatura de Introducción al Software en la que los alumnos puedan aplicar sobre un sistema embebido como es Micro:bit, los conceptos revisados en la asignatura.

2. Planificación y metodología seguidas

2.1. Ejecución de ejemplos en Microbian

El desarrollador del SO[1] solía mantener un repositorio con multitud de ejemplos de uso de las distintas funcionalidades disponibles en Microbian. Estos ejemplos resultaron de gran utilidad a la hora de conocer las funciones que más adelante servirían para implementar la librería de alto nivel, así como su funcionamiento a nivel del sistema operativo y a nivel del hardware.

Además, en esta fase de familiarización con el sistema operativo Microbian, se introdujo el concepto de la compilación cruzada.

2.2. Desarrollo de herramientas

Una vez revisados los ejemplos y entendidas las peculiaridades del proceso de compilación y carga, pudo procederse con el desarrollo de dos scripts que permitirían simplificar estos procesos.

2.3. Desarrollo de la librería de alto nivel

Ya con las herramientas terminadas y, por tanto, habiendo cubierto uno de los grandes objetivos del proyecto, podría dar comienzo el desarrollo de la librería de alto nivel, en primer lugar, identificando el hardware presente en la placa, y seleccionando aquellos dispositivos y sensores que pudieran resultar de especial interés, bien por las posibilidades que a priori ofrecían de cara a desarrollos futuros por parte de los alumnos, o bien por su vistosidad, es decir, lo llamativos que resultarían de cara a trabajar con ellos, así como su facilidad de uso.

2.4. Documentación y distribución

Tras completar el segundo gran objetivo del trabajo, y habiendo probado de una forma lo suficientemente exhaustiva el correcto funcionamiento de las funciones desarrolladas, podría generarse la documentación asociada a la librería, así como el paquete de software (*.deb*) con que se distribuirían tanto la librería como las herramientas, de forma que los alumnos pudieran disponer de todo lo necesario para trabajar sobre la placa de una manera rápida y sencilla.

[1] Sistema Operativo

3. Herramientas y tecnologías empleadas

3.1. Micro:bit

Se trata de una placa de desarrollo ideada inicialmente para el acercamiento de jóvenes y niños al mundo de la programación, pero que se espera pueda ser adaptada para su uso en la docencia de asignaturas en niveles superiores de educación, como se verá más adelante en el Capítulo 6, cuando se profundice en los detalles de la librería surgida como resultado del proyecto.

Es un hardware de muy bajo costo, abierto, y que soporta una gran variedad de add-ons con los que ampliar sus capacidades.

En su segunda versión, la cual fue la empleada durante todo el desarrollo del trabajo, cuenta con un microcontrolador ARM Cortex M4 de 32 bits (Nordic nRF52833) [3], 512 kB de ROM y 128 kB de RAM, que lo dotan de capacidad de cómputo y almacenamiento suficientes para ejecutar los proyectos que alguien que se inicia en la programación de sistemas embebidos pueda llegar a construir.

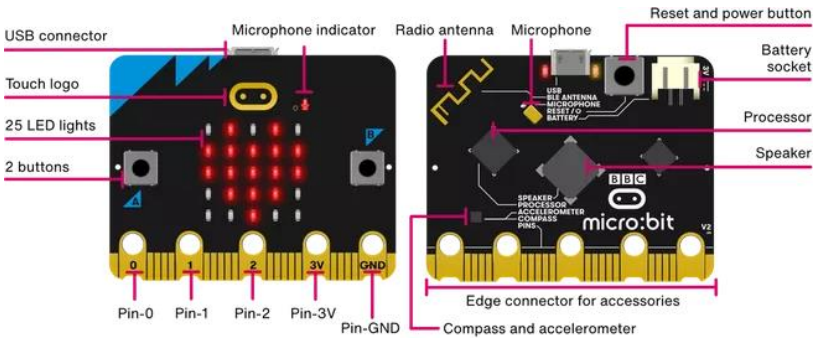

Figura 1: Micro:bit V2

Uno de los aspectos fundamentales que llevan a la elección de esta placa es el hecho de que integra, sin necesidad de circuitería externa, el siguiente conjunto de dispositivos y sensores, los cuales aparecen señalados en la Figura 1[2]:

- **GPIO:** la interfaz que ofrece el microcontrolador soporta la entrada/salida a través de los pines situados en la parte inferior de la placa, así como las comunicaciones con un subconjunto de los dispositivos que se encuentran integrados en ella.

- **Display:** una matriz de 5x5 LEDs de intensidad variable, y configurables a través de la interfaz del GPIO.

- **Botones:** cuenta con tres botones pulsadores, dos en la parte anterior y uno en la parte posterior. Este último actúa a modo de *reset* de la placa. También dispone de un botón táctil situado en el logo de la parte frontal, además de que los pines 2, 10 y 19 del GPIO, que en la placa aparecen numerados como 'P0', 'P1' y 'P2', pueden configurarse para detectar pulsaciones y actuar a modo de botones táctiles.

- **Sensor de movimiento:** se trata de un sensor de aceleración y un magnetómetro integrados en el mismo chip, que proporcionan, respectivamente, medidas de la intensidad de los campos gravitatorio y magnético en los tres ejes necesarios para describir el vector de estas magnitudes.

[2] Micro:bit Educational Foundation. (2023). *Nuevo micro:bit con sonido* [webp]. Micro:bit – Guía del usuario. https://microbit.org/es-es/get-started/user-guide/overview/

- **Sensor de temperatura:** se encuentra integrado en el propio microcontrolador y permite realizar medidas de la temperatura de este.

- **Zumbador piezoeléctrico:** se encuentra mapeado en la interfaz del GPIO y permite generar tonos de una frecuencia concreta, correspondiente a la frecuencia de oscilación de la membrana del dispositivo.

- **Comunicaciones inalámbricas:** el microcontrolador soporta comunicaciones a través de Bluetooth (BLE), así como un protocolo de comunicaciones por radio propietario de Nordic que opera en la misma banda [4].

- **Micrófono**

3.2. Microbian

Microbian es un sistema operativo embebido multiproceso desarrollado por el profesor Mike Spivey, cuyo propósito es utilizarlo como recurso didáctico en alguna de las asignaturas que imparte en la Universidad de Oxford.

Actualmente existen una gran variedad de librerías para programar sobre Micro:bit, y en un amplio abanico de lenguajes de programación como pueden ser Python, Javascript, Rust o Ada. Por ejemplo, la Universidad de Lancaster, la principal institución educativa que colabora con el proyecto Micro:bit, matiene una librería C/C++ con soporte para todo el hardware de la placa [5], y que sirve de soporte para numerosas librerías desarrolladas en otros lenguajes. No obstante, esta librería de alto nivel hace accesible toda su funcionalidad a través del lenguaje C++, además de que requiere la instalación de otros componentes adicionales para su uso (p.ej., ARM mbed o el Nordic SDK). Esta complejidad hace que esta librería no sea considerada ideal para ser usada en una asignatura introductoria como "Introducción al Software".

Micro:bit también puede programarse desde el IDE de Arduino, puesto que cuenta con soporte para la placa a través de la instalación de un plug-in, pero la instalación del plug-in por sí mismo no nos proporcionará funciones de alto nivel con las que poder interactuar con el hardware de la placa, y la única librería con un adecuado soporte y que podría proporcionarnos tales funcionalidades está implementada en el lenguaje C++ [6].

En definitiva, todas las alternativas revisadas, o bien están enfocadas a la programación en C++, o bien no proporcionan toda la funcionalidad de cabría esperar, o simplemente se trata de drivers concretos para un único dispositivo. Es por ello por lo que se consideró que Microbian sería el mejor candidato a ser el punto de partida sobre el que desarrollar una API con la que los alumnos puedan programar la plataforma Micro:bit en el lenguaje C y, dado que este sistema operativo es la base sobre la que se construye el proyecto, indicar que se explicarán los detalles de su funcionamiento interno en el Capítulo 4.

3.3. Git

Desarrollado inicialmente por Linus Torvalds como una herramienta para el mantenimiento organizado del desarrollo del kernel de Linux, puede decirse que hoy en día es el más popular sistema de control de versiones de código abierto, y el elegido para el desarrollo de este proyecto[3].

[3] El repositorio del proyecto puede encontrarse en la siguiente dirección web: https://github.com/eon0111/Micro-bit-C-API.git

3.4. Doxygen

La documentación de cualquier proyecto y, concretamente, la de un proyecto software, resulta imprescindible para garantizar la mantenibilidad de dicho proyecto y, por tanto, de todos y cada uno de los desarrollos software que lo constituyan.

Doxygen es un aplicativo que posibilita la generación automática de la documentación de un fichero de código fuente, o incluso un proyecto software completo, a partir de un fichero de configuración que defina todas las propiedades de la documentación resultante, y haciendo uso de un sistema de etiquetas en comentarios en el código con las que designar las características de este que quieran documentarse.

Desarrollado en 1997 por Dimitri Van Heesch [7], soporta multitud de lenguajes de programación tales como C++, Java, Python y, por supuesto, C, además de que permite la generación de documentación en múltiples formatos de salida (HTML, LaTeX, PDF, etc.), lo que lo convierte en una herramienta muy versátil, sencilla y de gran utilidad a la hora de documentar soluciones software, además de ser el sistema de documentación empleado en la asignatura de Introducción al Software, y en este proyecto, siguiendo el estilo empleado en la documentación del asignatura.

Toda la documentación generada puede encontrarse en el repositorio del proyecto.

3.5. ARM GNU Toolchain

La compilación cruzada es el proceso mediante el cual se genera un ejecutable que pueda ejecutar, valga la redundancia, sobre una arquitectura hardware distinta de la arquitectura de la máquina donde se originó dicho ejecutable. Esto resulta de gran utilidad a la hora de compilar soluciones software para dispositivos cuyas capacidades de cómputo y/o almacenamiento no permitan realizar dicha compilación en ese mismo dispositivo, o quizá no permitan hacerlo de una forma ágil. En la Figura 2 se proporciona un esquema con una simplificación del proceso de compilación cruzada.

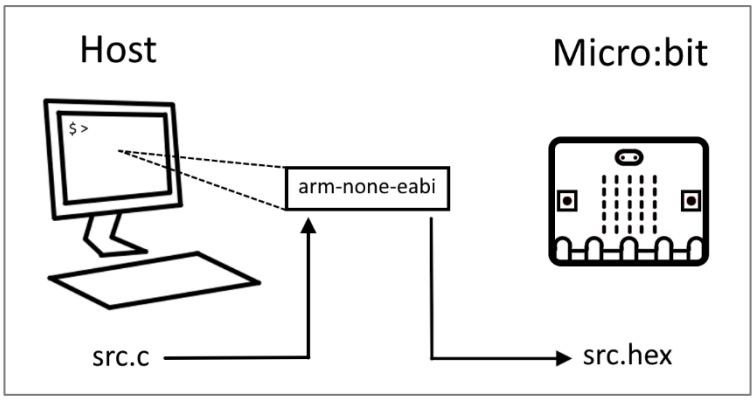

Figura 2: Esquema del proceso de compilación cruzada

Debe tenerse en cuenta que el microcontrolador que monta la placa posee una arquitectura ARM de 32 bits, por lo que se hace mandatorio utilizar un compilador cruzado con el que poder generar ejecutables con el formato e ISA[4] adecuados, de ahí que deba emplearse un compilador cruzado como es *arm-none-eabi-gcc*. Este compilador es mantenido por la Free Software

[4] Del inglés *Instruction Set Architecture*, es el conjunto de instrucciones que una arquitectura hardware puede entender y ejecutar

Foundation como parte del proyecto GNU GCC, y dota a sus usuarios de capacidad para generar ejecutables que puedan correr sobre un amplio abanico de microcontroladores basados en la arquitectura ARM. Por defecto genera ejecutables en el formato ELF, el cual está pensado principalmente para ejecutar sobre sistemas operativos Unix-like, y no de forma baremetal sobre Micro:bit, de ahí que debamos hacer uso de la utilidad *arm-none-eabi-objcopy* para obtener un fichero ejecutable en formato HEX que pueda ejecutar sin problemas en la placa.

3.6. Protocolo I2C

Es uno de los múltiples protocolos que pueden emplearse en las comunicaciones entre el microcontrolador y los periféricos, haciendo uso de únicamente dos conectores, uno para la transmisión de datos (SDA) y, el otro (SCL), para la transmisión de la señal de reloj con la que maestro y esclavo podrán sincronizarse. Puede verse en la Figura 3 un esquema con el conexionado básico entre el dispositivo *master* y los dispositivos *slave* I2C.

En el contexto de este proyecto, se emplea este protocolo en el manejo del sensor de movimiento, puesto que se encuentra conectado al bus I2C interno de la placa. A demás, y a pesar de no haberle dado soporte en la librería generada, dos de los pines presentes en la placa pueden emplearse para las comunicaciones a través del bus I2C externo.

Tal y como se detalla en la Figura 4, en una comunicación I2C, se suceden los siguientes eventos:

1. El dispositivo maestro establece en el bus la condición de comienzo de la comunicación, y envía la dirección del dispositivo con el que quiere establecer dicha comunicación, seguida de un bit indicando la operación que solicita (lectura o escritura en el dispositivo).
2. Esta dirección es recibida por todos los dispositivos "esclavos" conectados al bus, pero tan sólo aquel cuya dirección coincida con la solicitada por el "maestro" le responderá con un ACK.
3. El maestro envía o recibe entonces el/los dato(s) en tramas de 1 Byte de longitud, todas seguidas de su correspondiente confirmación, por parte del maestro cuando este recibe datos, o por parte del esclavo cuando el primero los envía.
4. La comunicación finaliza cuando el maestro establece en el bus la condición de parada.

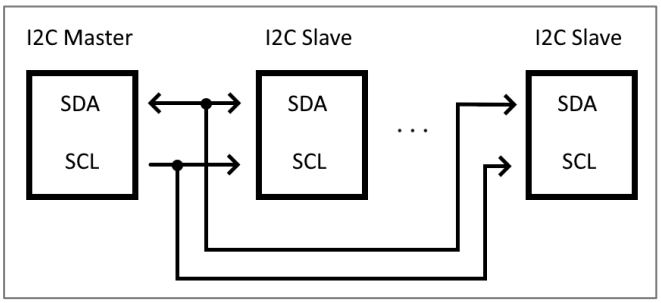

Figura 3: Esquema de conexiones en el bus I2C

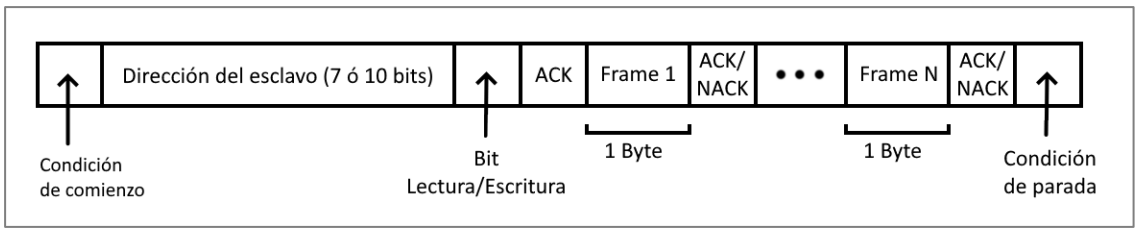

Figura 4: Esquema de comunicaciones I2C

3.7. GPIO

GPIO son las siglas de General Purpose Input/Output, y hace referencia al conjunto de pines que pueda poseer una placa de desarrollo o cualquier circuito integrado en general, los cuales no han sido diseñados e instalados en dicho hardware con un propósito concreto, si no que permiten al usuario generar señales de salida mediante la aplicación de determinados niveles de tensión en los pines, así como leer las posibles señales de entrada que puedan llegar a estos.

Como puede verse en la Figura 5[5], la placa dispone de 20 pines que pueden ser utilizados como entrada/salida genéricas, además de una interfaz GPIO interna donde se conectan parte de los dispositivos hardware integrados en el Micro:bit.

En el contexto de este proyecto, el zumbador, el display y los botones han podido ser accedidos por medio de la interfaz GPIO interna.

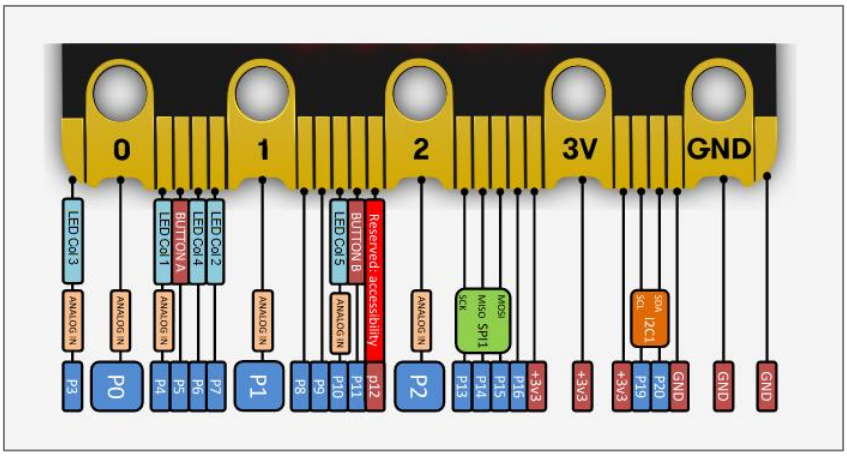

Figura 5: Esquema de los pines de la placa. En azul con tipografía en color blanco, los pines destinados a la entrada salida genérica a través de la interfaz GPIO externa

[5] Micro:bit Educational Foundation. (2023). *Edge Connector Pins* [svg]. Edge Connector & micro:bit pinout. https://tech.microbit.org/hardware/edgeconnector/

4. Microbian

El punto de partida de este trabajo, como se comentaba en el apartado 3.2, ha sido el sistema operativo Microbian, el cual es un sistema operativo multiproceso, basado en la primitiva de sincronización a través de paso de mensajes. Este SO integra drivers para el manejo del siguiente hardware:

- Matriz de LEDs
- Interfaces I2C interna y externa
- Comunicaciones inalámbricas a través del protocolo propietario de Nordic en la banda de 2.4 GHz
- Interfaz serie
- Interfaces GPIO interna y externa

No obstante, los drivers disponibles permiten la interacción con el hardware en un nivel de abstracción no lo suficientemente alto como para que un alumno que se inicia en la programación en lenguaje C pueda centrarse únicamente en asimilar los conceptos inherentes a dicho lenguaje, obviando la forma de operar con el hardware, sin tener que dedicarle tiempo adicional al aprendizaje de protocolos, modos de operación, etc.

Además, algunos dispositivos y sensores no contaban con soporte directo por parte de Microbian, por lo que fue necesario incorporar a la librería Ubit las funcionalidades que permitiesen su manejo de una forma sencilla. En lugar de proporcionar el conjunto de llamadas con que interactuar con cada dispositivo concreto, el sistema operativo se limita a proporcionar las funciones con que operar con las interfaces a las que dichos dispositivos se encuentran conectados, bien sea la interfaz GPIO interna del microcontrolador, o la interfaz I2C.

Sobre el manejador que implementa Microbian para la interfaz I2C, comentar que, tras la inicialización de la interfaz, el sistema operativo lanza un proceso que continuamente se encontrará a la espera de mensajes provenientes del resto de procesos cuando estos deseen establecer alguna comunicación, bien sea de lectura o escritura, a través del bus I2C interno o externo. Además, y de un modo similar, el sistema operativo mantiene un proceso interno encargado de realizar un refresco en la matriz de LEDs, de tal modo que de la apariencia de estar mostrándose en todo momento una imagen completa en el display, cuando en realidad se está recorriendo una variable del sistema operativo la cual alberga el estado de cada LED individual.

En Microbian el punto de entrada a los códigos del usuario se encuentra en una función de nombre *init*, la cual debe ser escrita por el usuario para cada aplicación, y cuyo prototipo se encuentra definido en el fichero *startup.c*. Cuando finaliza el arranque del sistema operativo, el proceso de arranque llama a esta función *init*, desde la cual podrán crearse los distintos procesos de usuario.

5. Herramientas de apoyo a la programación en la placa Micro:bit

Las dos operaciones básicas que los alumnos deberían ser capaces de realizar para poner a prueba sus códigos sobre la placa son la compilación y la carga del ejecutable resultante en la memoria *flash* del dispositivo. Es por ello por lo que, junto con la librería, se proporcionan dos scripts desarrollados en *bash*, con los que podrán llevar a cabo las operaciones descritas.

5.1. Compilación (ubit-gcc)

La obtención de ejecutables para la arquitectura del microcontrolador ARM de la placa no es un proceso tan trivial como pueda llegar a ser el compilar un código fuente que ejecute sobre el mismo sistema operativo en que se desarrolló dicho código. Es necesario simplificar el proceso de compilación, proporcionando una experiencia de uso similar, si no idéntica, a la que se obtendría con la ejecución del compilador GCC sobre los códigos que habitualmente se desarrollan en la asignatura de Introducción al Software en una plataforma Linux (esto es, gcc -o <binario> <fuente.c> <librerías externas>).

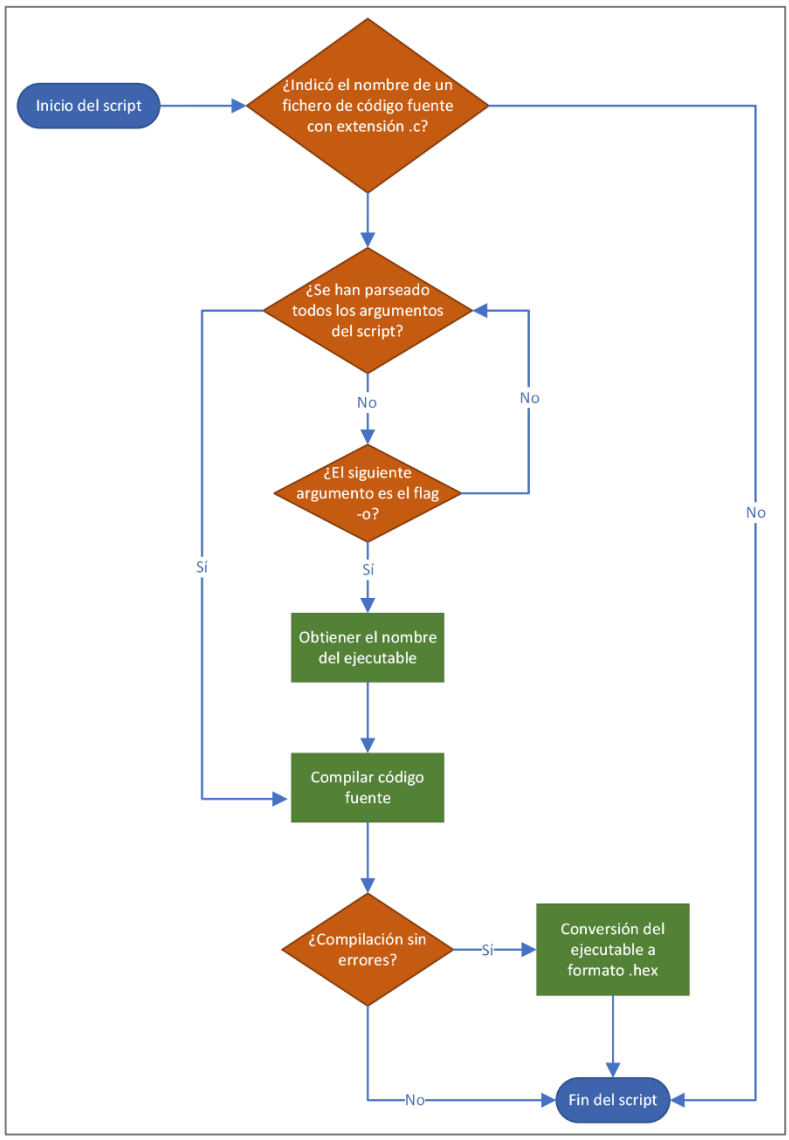

Figura 6: Flujo de ejecución del script *ubit-gcc.sh*

Tal y como se ilustra en la Figura 6, el script comprobará, en primer lugar, que, en el momento de ejecutarlo desde el terminal, se ha indicado un fichero de código fuente con extensión ".c" puesto que, de no hacer esta comprobación, más adelante cuando se invoca al compilador con todos los argumentos pasados al script, el linker mostraría el error presentado en la Figura 7, indicando que la función main, que es el punto de entrada al código del usuario de la librería Ubit, no se encuentra definida. Se evita este error de difícil comprensión para facilitar al alumno (que con cierta probabilidad será lego en la materia) la detección del error cometido.

El script, del cual se proporciona un ejemplo de uso en la Figura 8, comprobará además el nombre del ejecutable que el usuario haya indicado, de forma que, de haber indicado un nombre a través del flag "-o", este sea modificado para contener la extensión ".hex". De lo contrario, en el momento de cargar el código en la placa, esta mostraría en el display el error 504, el cual puede indicar o bien que ha expirado el tiempo disponible para realizar la transferencia del fichero, o bien que, efectivamente, el fichero no cuenta con la extensión adecuada [8].

Las opciones empleadas de manera interna (y transparente para el usuario) por el script en la herramienta de compilación cruzada son las siguientes[9]:

- **-mcpu=cortex-m4:** el nombre de la CPU sobre la que ejecutará el binario. Permite al compilador conocer tanto el nombre de la arquitectura como el tipo de procesador, de forma que se genere código binario para esa arquitectura y microprocesador concretos.

- **-mthumb:** el ISA al que deberán pertenecer las instrucciones generadas. Puede generarse un ejecutable con instrucciones del tipo Thumb o ARM, siendo las primeras un subconjunto con codificaciones más compactas de 16 bits de las segundas, lo que permite generar binarios más compactos.

- **-O2:** el nivel de optimización que deberá aplicar en la compilación.

- **-ffreestanding:** la librería estándar podría no existir en el entorno en que ejecute el binario, y el punto de entrada del programa podría no ser un "main".

- **-I /usr/local/include:** el directorio donde el compilador deberá buscar los ficheros de cabecera tanto de la librería *Ubit* como de Microbian.

- **-T /usr/share/ubit/nRF52833:** el script de enlazado.

- **-nostdlib:** indica al compiladro que no deberá enlazar el código con las librerías estándar del sistema, puesto que se emplearán las funcionalidades provistas por Microbian

- **-lgcc -lc:** indica al compilador que sí deberá enlazar el código con las librerías estándar de gcc (*libgcc*) y C (*libc*), las cuales serían de otro modo pasadas por alto al haber indicado el flag *-nostdlib*.

Para obtener un fichero ejecutable en el formato que acepta Micro:bit se hace uso de la herramienta *arm-none-eabi-objcopy*, a la cual se indica el formato de salida al que debe convertir el fichero *.elf* que genera por defecto la herramienta de compilación. El fichero *.hex* resultante es un binario que contiene las instrucciones a ejecutar "en crudo".

```
[ [redacted] ] : [redacted]/Micro-bit-C-API/herramientas > (desarrollo)
$ ubit-gcc -o prueba-display
/usr/lib/gcc/arm-none-eabi/8.3.1/../../../arm-none-eabi/bin/ld: /usr/share/ubit/libubit.a(misc.o): in function `init':
/home/eon/Documentos/TFG/Micro-bit-C-API/ubit/misc.c:80: undefined reference to `main'
collect2: error: ld returned 1 exit status
```

Figura 7: Error en el script cuando no se comprueba el fichero de código fuente

```
$ ubit-gcc -o gameoflife gameoflife.c
$ ll
total 68K
drwxr-xr-x 2 eon eon 4,0K jul  2 00:10 .
drwxr-xr-x 5 eon eon 4,0K jul  2 00:09 ..
-rw-r--r-- 1 eon eon 2,2K jul  2 00:10 gameoflife.c
-rw-r--r-- 1 eon eon  49K jul  2 00:10 gameoflife.hex
```

Figura 8: Ejemplo de ejecución del script de compilación

Por último, esta herramienta también admite que los alumnos utilicen sus propios flags de compilación (p.ej., para activar ciertos avisos en la compilación con -Wall).

5.2. Carga de ejecutables (ubit-load)

Con respecto al proceso de carga de ejecutables en la placa, este resulta muy sencillo cuando se realiza desde una interfaz gráfica, puesto que es posible copiar y pegar el binario en el sistema de ficheros que el sistema operativo monta de manera automática cuando se conecta la placa al equipo. No obstante, este proceso puede tornarse más complejo cuando se realiza desde un terminal de texto.

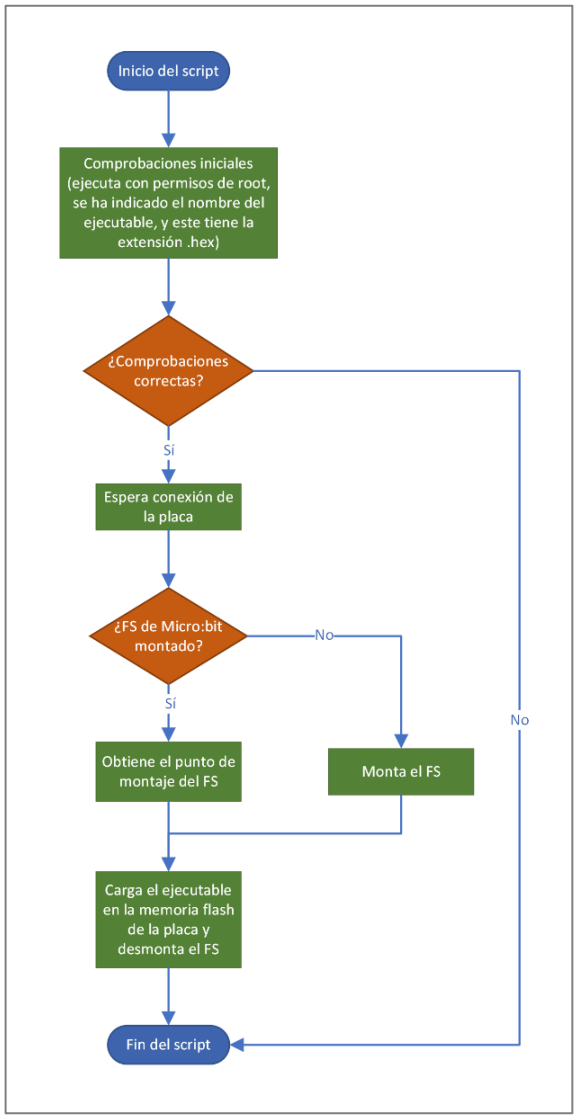

Figura 9: Flujo de ejecución del script *ubit-load.sh*

Además de la complejidad inherente al proceso de carga de código por medio del terminal, existe la particularidad comentada en el apartado anterior, de que los ejecutables que quieran cargarse deben tener en sus nombres la extensión ".hex" para no generar un error en el proceso de carga. Para automatizar este proceso, evitando así que los alumnos deban hacer frente a los dos problemas comentados, se ha creado el script *ubit-load*, cuya funcionalidad se muestra en el diagrama de flujo de la Figura 9.

Se ha procurado automatizar este proceso en la medida de lo posible, haciendo que los ejecutables se copien en la placa una vez esta ha sido conectada al equipo. Una vez conectada, y habiendo comprobado que el fichero a cargar posee la extensión adecuada, se monta el sistema de ficheros del Micro:bit en un directorio temporal donde se copia el ejecutable y, una vez copiado, se desmonta el sistema de ficheros y se elimina el punto de montaje. Todo ello da como resultado un script cuyo uso resulta de una gran sencillez, como puede verse en el ejemplo mostrado en la Figura 10.

Figura 10: Ejemplo de ejecución del script de carga

6. La librería Ubit

Tras haber realizado una selección inicial del hardware al que se le daría soporte a través de la librería, la cual recibe el nombre de "Ubit", se definió el conjunto de funciones que la conformarían, así como su forma, es decir, el formato que deberían seguir sus nombres y los argumentos que recibirían.

Como resultado del desarrollo de esta librería de alto nivel se ha generado la arquitectura mostrada en la Figura 11, donde la librería Ubit hace uso de los manejadores proporcionados por Microbian para ofrecer a los usuarios una visión simplificada de los dispositivos y sensores disponibles en la placa Micro:bit, además de que proporciona nuevos manejadores para aquellos dispositivos y sensores que actualmente no están soportados de manera directa por Microbian.

Como se muestra en la Figura 11, la librería contiene, por un lado, el fichero de cabecera principal (ubit.h), el cual alberga los prototipos de todas las funciones implementadas para el manejo de los distintos dispositivos, así como las definiciones de algunos enumerados y nuevos tipos de datos. Las cabeceras de estas funciones pueden consultarse en el *Anexo A*.

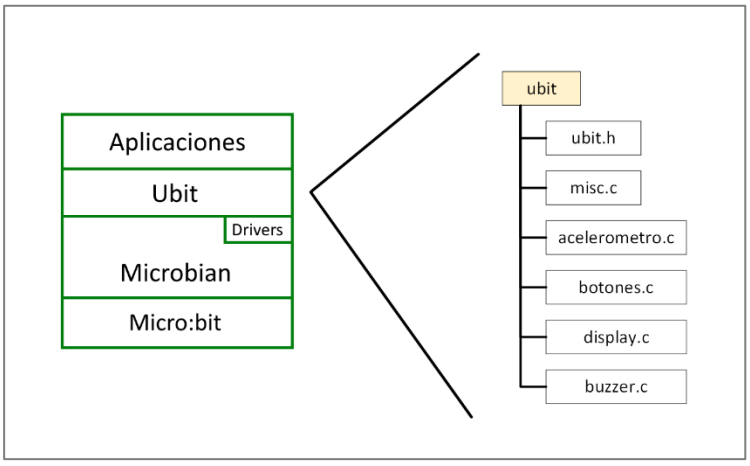

Figura 11: Diagrama con las capas de la arquitectura y detalle de los ficheros de que se compone la librería Ubit

Por otro lado, y en lugar de generar un único fichero de código fuente con todas las implementaciones de alto nivel para el manejo de todos los dispositivos, se decidió, con objeto de mantener una mejor organización de la librería, segregar las implementaciones en distintos ficheros, cada uno dedicado al manejo de un dispositivo concreto. Así, se ha dado soporte a los siguientes dispositivos y sensores, de los cuales tan sólo la matriz de LEDs contaba con un driver específico en Microbian, cuya funcionalidad se ha simplificado y ampliado:

- Sensor de movimiento, en *acelerometro.c*
- Botones (de pulsación y táctiles), en *botones.c*
- Zumbador piezoeléctrico, en *buzzer.c*
- Display LED, en *display.c*
- Sensor de temperatura, en *misc.c*

Con respecto al nombrado de las funciones, y con objeto de mejorar la usabilidad de la librería, se optó por darles un nombre que siguiera el esquema "[dispositivo]_[acción]", el cual es fácilmente memorizable y muy intuitivo. Además, todas aquellas funciones que reciban parámetros los cuales requieran de una comprobación, retornan el valor 0 o -1 en función de si la función tuvo una terminación correcta o incorrecta, respectivamente. También es importante

destacar que, en lo referente a los argumentos, se decidió evitar en la medida de lo posible el paso por referencia, puesto que los punteros conforman una parte pequeña y emplazada más hacia el final del temario de la asignatura de Introducción al Software. Por otro lado, y con respecto a la definición de los tipos de datos, sus nombres siguen el criterio empleado en la asignatura de Introducción al Software, es decir, *[nombre]_t*.

Por último, se ha procurado abstraer a los alumnos de las peculiaridades del sistema operativo, requiriendo por su parte tan solo que los códigos que desarrollen guarden la estructura mostrada en la Figura 12. Simplemente deberán incluir el fichero de cabecera de la librería para poder hacer uso de las funciones desarrolladas, así como definir una función principal de nombre *main*, puesto que la función *init* ya se encontrará definida en el fichero *misc.c*, y en ella se realizará la inicialización de todo el hardware soportado tanto por Microbian como por la librería Ubit, además de que se creará el proceso que ejecutará el código definido en la función principal del usuario, la cual deberá recibir ese nombre concreto.

En definitiva, la librería proporciona funciones y tipos de datos asequibles para alumnos sin conocimientos de programación previos.

```
#include <ubit.h>

void
main (int n)
{
    microbit_inicializa_hardware();
    // TODO
}
```

Figura 12: Estructura de un programa

6.1. Soporte a la inicialización del hardware

Es necesario realizar una configuración inicial del hardware antes de hacer uso de este, estableciendo los parámetros iniciales que determinarán el comportamiento de los sensores y dispositivos a los que la librería Ubit da soporte. Esta inicialización se lleva a cabo dentro de la función *init*, definida en el fichero *misc.c*. Se inicializa el timer para que posteriormente la tarea de refresco del display pueda realizarlo con la periodicidad con que está programada. También se inicializan las interfaces I2C y el display, además de que se construye una imagen vacía sobre la variable de la librería Ubit (*imagen_actual_microbian*) definida en *ubit.h*, que mantiene el estado de la imagen que se mostrará en la matriz de LEDs. Asimismo, se configuran como entradas los pines del GPIO a los cuales se encuentran conectados los botones, táctiles y de pulsación, así como también se configura, esta vez como salida, el pin del GPIO al que se conecta el zumbador piezoeléctrico.

Durante el desarrollo de esta función *init* surgió un problema con la inicialización del acelerómetro, y es que este no pudo ser inicializado desde esta función puesto que ello provocaba un *kernel panic* en Microbian. Esta situación es señalada por el sistema operativo por medio de una imagen concreta en el display, el cual permanece en una intermitencia constante hasta que se reinicia la placa. Fruto de este error es que se desarrollara la función *microbit_inicializa_hardware*, la cual debe ser llamada desde el código del usuario antes de hacer uso de cualquiera de las funciones con las que interactuar con el acelerómetro. Se le ha dado este nombre genérico para no llevar a confusión y que los usuarios de la librería piensen que deben inicializar todo el hardware antes de utilizarlo.

A diferencia de la compilación nativa, en la que el punto de entrada al código del usuario se encuentra en la función *main*, en Microbian esta función recibe

En resumen, y tal y como aparece indicado en la Figura 13, cuando finaliza el arranque del sistema operativo, el proceso encargado de esa tarea realiza una llamada la función *init*, que en condiciones normales sería el punto de entrada al código del usuario. No obstante, y para facilitar la programación del hardware soportado por la librería Ubit, dentro de esa función *init* se realiza la inicialización de dicho hardware, y se crea un proceso el cual ejecutará el código definido por el usuario en una función de nombre *main*.

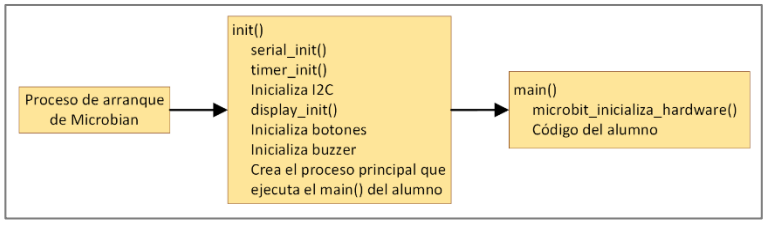

Figura 13: Secuencia de arranque hasta llegar al código del alumno

6.2. Manejador del sensor de movimiento

Microbian no proporciona un driver específico para el manejo del sensor de movimiento, si no el conjunto de funciones con que operar sobre los buses I2C interno y externo. El sistema operativo mantiene un proceso interno cuyo propósito es la recepción de mensajes provenientes del resto de procesos cuando estos realizan llamadas a las funciones de lectura o escritura sobre el bus I2C.

Para realizar la lectura de algún registro en un dispositivo conectado y mapeado sobre el bus I2C de la placa, se deposita en el registro de dirección (ADDRESS [10]) del controlador (maestro) I2C la dirección del dispositivo al que quiera accederse, y se manda esa dirección a través del bus, escribiendo el valor '1' en el registro TASKS_STARTTX, con lo que se inicia la comunicación. Tras el envío de la dirección del dispositivo a través del bus podrá procederse con el envío de la dirección del registro del dispositivo cuyo valor quiera leerse, tarea que realiza el proceso de Microbian encargado del manejo de las comunicaciones I2C, el cual comenzará a recibir los datos solicitados al dispositivo una vez haya activado la recepción en el registro EVENTS_RXDREADY del maestro.

6.2.1. Inicialización

El primer paso para manejar el acelerómetro es inicializarlo por medio de la función mostrada en la Figura 14, la cual realiza la escritura de los valores de configuración apropiados en el registro de control del dispositivo (CTRL_REG1_A). Se trata de un registro de 8 bits en el que los 4 más significativos establecen la frecuencia de muestreo del sensor y, los 3 menos significativos, activan o desactivan la medición en cada uno de los 3 ejes disponibles, resultando la escritura del valor '1' sobre la posición correspondiente a cada eje en la activación de dicho eje y, un '0', en la desactivación del eje [11]. El bit restante activa o desactiva el modo de bajo consumo, en el cual las lecturas de la aceleración tienen una resolución de 8 bits [12].

```
#define ACC             0x19
#define ACC_CTRL_REG1   0x20

void acelerometro_inicializa()
{
    i2c_write_reg(I2C_INTERNAL, ACC, ACC_CTRL_REG1, 0x4f);
}
```

Figura 14: función para la inicialización del acelerómetro

La constante de Microbian I2C_INTERNAL, la cual es pasada como primer argumento de la función *i2c_write_reg* como puede verse en la Figura 14, le indica a esta función el bus I2C (interno o externo) al que se encuentra conectado el dispositivo, ACC es la dirección que recibe el acelerómetro en el bus interno de la placa, y ACC_CTRL_REG1 es la dirección del registro de control del dispositivo.

En la función *acelerometro_inicializa*, el valor escrito sobre el registro de control del acelerómetro, y descrito en la Figura 15, resulta en la activación de mediciones en todos los ejes, así como la activación del modo de bajo consumo. También se establece la frecuencia de muestreo a 50 Hz, siguiendo las indicaciones proporcionadas en la especificación del dispositivo con respecto a la configuración del ODR (Output Data Rate) [13].

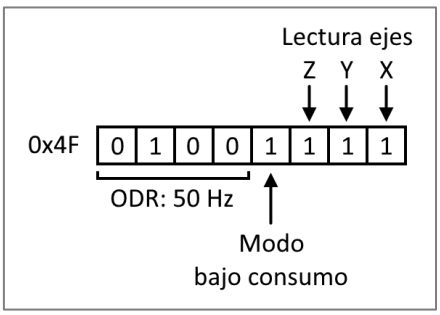

Figura 15: Valor de inicialización del acelerómetro

6.2.2. Lectura de valores de aceleración

Se proporcionan funciones para realizar la lectura en bruto de los datos depositados en el registro de datos de cada eje, así como dos funciones para obtener el valor de inclinación (en grados) de la placa respecto de los ejes X e Y.

```
#define ACC_OUT_X       0x29
#define ACC_OUT_Y       0x2B
#define ACC_OUT_Z       0x2D

int
acelerometro_lectura_x()
{
    signed char tmp;
    i2c_read_bytes(I2C_INTERNAL, ACC, ACC_OUT_X|0x80, (byte *) &tmp, 1);
    return tmp;
}
```

Figura 16: función para la lectura del valor de aceleración en el eje X

Las tres funciones desarrolladas, una de las cuales se muestra en la Figura 16, son idénticas salvo por la dirección del registro a leer que se indica a la función *i2c_read_bytes*. En estas funciones se retorna el byte leído de los registros de datos del acelerómetro, cuyas direcciones son las definidas en las constantes ACC_OUT_X/Y/Z, siendo el rango de valores posibles para las lecturas [-128, 128], y 64 el valor obtenido en reposo, correspondiente al valor de la aceleración de la gravedad sobre la superficie terrestre.

Se decidió proporcionar a los alumnos las funciones de lectura de datos en bruto puesto que un ejercicio interesante podría ser la implementación de las funciones de cálculo de la inclinación, una de cuyas posibles implementaciones se detalla en el apartado siguiente.

6.2.3. Inclinación

Siendo conocedores del valor de intensidad del campo gravitatorio terrestre en cada uno de los tres ejes *X*, *Y* y *Z*, puede calcularse la inclinación del dispositivo respecto de la horizontal, es decir, la línea perpendicular a aquella que une el dispositivo con el centro de masas del planeta. Para ello puede hacerse uso de la definición de la tangente, la cual relaciona los dos catetos de un triángulo rectángulo del siguiente modo:

$$\tan\alpha = \frac{Cateto\ Opuesto}{Cateto\ Contíguo} = \frac{G_z}{G_{x\ ó\ y}}$$

Siendo $G_i, i \in \{x, y, z\}$ el valor de la aceleración en cada eje.

A partir de esta expresión, puede obtenerse el valor del ángulo con la función inversa de la tangente, el arcotangente:

$$\alpha = \tan^{-1}\frac{G_z}{G_{x\ ó\ y}}$$

Dado que Microbian no implementa ninguna función que permita realizar las operaciones matemáticas necesarias para obtener el cálculo de la inclinación, es necesario añadir estas funciones en la librería y, dado que tan solo van a emplearse en el cálculo de la inclinación, pueden definirse de manera estática[6] en el driver del acelerómetro.

A continuación, en la Figura 17, se detalla la implementación realizada de la aproximación en series de Taylor de la función arcotangente.

```
#define PI 3.14159265358979323846
#define RAD_A_GRAD 180 / PI   /* = 360 / (2 * PI) */
#define abs(x) (x < 0) ? -x : x

static float
atan(float x)
{
    int n;
    float x2 = abs(x), r = 0.0;
    float c = (x2 < 1) ? 1 : -1, c2 = (x < 0) ? -1 : 1;

    for (n = 0; n < 10; n++)
        r += pow(-1,n) * pow(x2,c*(1+2*n)) / (1+2*n);

    return (x2 < 1) ? c2 * r : c2 * ((PI / 2) - r);
}
```

Figura 17: Implementación de la aproximación en series de Taylor de la función arcotangente

La función arcotangente retorna el resultado en radianes, lo cual a priori podría no resultar tan útil como una medida de la inclinación en grados, por lo que el dato se retorna ya convertido, y en el rango [-90, 90]. Se proporciona en la Figura 18 la implementación de las funciones donde

[6] En el lenguaje C, el modificador *static* antepuesto a la declaración de una función hace que dicha función sea visible tan solo dentro del fichero de código fuente donde fue declarada

se realiza el cálculo de la inclinación y, en la Figura 19, la disposición de los ejes sobre la placa, así como los rangos de movimiento contemplados.

```
float
acelerometro_inclinacion_eje_x() {
    float d = atan((float)acelerometro_lectura_z() /
                   (float)acelerometro_lectura_x()) * RAD_A_GRAD;
    return (d < 0) ? -90 - d : 90 - d;
}

float
acelerometro_inclinacion_eje_y() {
    float d = atan((float)acelerometro_lectura_z() /
                   (float)acelerometro_lectura_y()) * RAD_A_GRAD;
    return (d < 0) ? -90 - d : 90 - d;
}
```

Figura 18: Funciones para el cálculo de la inclinación de la placa en los ejes X e Y

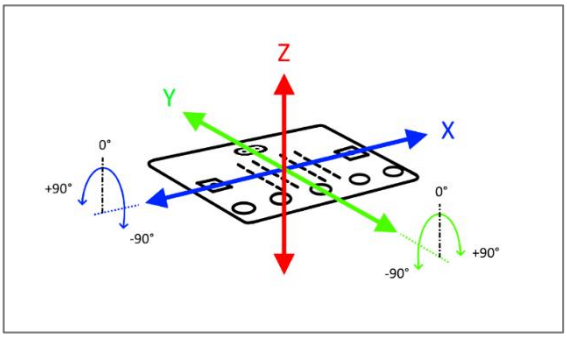

Figura 19: Disposición de los ejes sobre la placa y rangos de inclinación contemplados

6.3. Manejador de los botones

Se ha dotado a la librería de funcionalidad para el manejo de los dos botones de pulsación presentes en el frontal de la placa, los cuales se encuentran conectados a los puertos 14 y 23 del puerto GPIO 0, así como el botón táctil en el logo (puerto 4 en el GPIO 1) y los botones táctiles situados en la hilera de conectores de expansión (puertos 2, 3, y 4 del GPIO 0) [14]. Todos estos dispositivos son accesibles a través del GPIO, de nuevo, sin soporte directo por parte del sistema operativo, pero sí a través de las funciones que ofrece para las comunicaciones con las interfaces GPIO del microcontrolador. Gracias a estas funciones es posible realizar lecturas y escrituras en los registros de datos y control de la interfaz, con los que poder manejar los dispositivos que se encuentran conectados a esta.

Se proporcionan dos funciones genéricas a través de las cuales poder interactuar con cualquiera de los botones, y cuyas implementaciones son mostradas en la Figura 22: una función (*boton_pulsado*), con la que poder consultar el estado de un botón (pulsado o no) en el momento de la llamada a la misma, y otra (*boton_espera_pulsacion*) con la que esperar, como su nombre indica, a que se pulse un botón concreto. Además, y dado que originalmente Microbian no implementaba ninguna funcionalidad para el manejo de los botones táctiles, fue necesario adaptarla para contener sus identificadores en el fichero *hardware.h*, quedando este de la manera mostrada en la Figura 20. PAD0, 1 y 2 son macros de Microbian que generan los identificadores de estos dispositivos, los cuales se encuentran conectados, respectivamente a los pines 2, 3 y 4

del puerto GPIO 0. Por ejemplo, PAD0 se extiende a DEVPIN(0,1), que hace un shift de 5 bits hacia la izquierda del índice del GPIO, y suma al valor restante el índice del pin.

```
#define DEVPIN(p, i) ((p<<5) + i)

#define PAD0  DEVPIN(0, 2)
#define PAD1  DEVPIN(0, 3)
#define PAD2  DEVPIN(0, 4)

#define TOUCH_BUTTON_LOGO DEVPIN(1,4)
#define TOUCH_BUTTON_0 PAD0
#define TOUCH_BUTTON_1 PAD1
#define TOUCH_BUTTON_2 PAD2
```

Figura 20: Definición de los identificadores de los botones en el formato que emplea internamente Microbian, y de las macros que los generan

Se proporciona un tipo de dato con el que indicar, como puede verse más adelante en la Figura 21, el botón cuyo estado quiera conocerse. Aunque en un principio los alumnos puedan no contar con conocimientos sobre la definición de tipos en el lenguaje C, se consideró que no les costaría asumir que las funciones pueden recibir una serie de palabras reservadas como argumentos.

```
typedef enum {BOTON_A, BOTON_B, BOTON_LOGO, BOTON_0, BOTON_1, BOTON_2} boton_t;
```

Figura 21: Definición del tipo de dato boton_t en el fichero de cabecera *ubit.h* de la librería

Para evitar más modificaciones en el código fuente del sistema operativo, se decidió mantener en el driver de alto nivel un array con los identificadores de los botones, indexados según el tipo de dato enumerado *boton_t*.

La función *boton_pulsado* recibe como argumento el dato de tipo *boton_t* que indica el botón cuyo estado quiere conocerse, y realiza dicha consulta a través de la función del sistema operativo *gpio_in*, a la que se pasa como argumento el identificador del dispositivo, empleando el formato usado a nivel del sistema operativo, es decir, con el quinto bit más significativo indicando el puerto GPIO, y los cuatro menos significativos indicando el pin dentro de dicho puerto.

```
static unsigned int botones[] = {BUTTON_A, BUTTON_B, TOUCH_BUTTON_LOGO,
                                 TOUCH_BUTTON_0, TOUCH_BUTTON_1,
                                 TOUCH_BUTTON_2};

int
boton_pulsado(boton_t b)
{
  return (b < BOTON_A || b > BOTON_2) ? -1 : !gpio_in(botones[b]) & 0x1;
}

int
boton_espera_pulsacion(boton_t b)
{
  if (b < BOTON_A || b > BOTON_2) return -1;

  while (!boton_pulsado(b)) timer_delay(100);

  return 0;
}
```

Figura 22: Implementación de las funciones para el manejo de los botones

Según se observó en las pruebas realizadas, las interfaces GPIO ponen los bits del registro de estado (IN) a 1 cuando los botones se encuentran en reposo y, a 0, cuando estos están siendo pulsados. Es por ello por lo que en *boton_espera_pulsacion* se retorna el valor leído del registro, pero habiéndole aplicado una negación bit a bit y una máscara para obtener el LSB.

Internamente, la función de Microbian *gpio_in*, cuya implementación se proporciona en la Figura 23, emplea la macro GET_BIT, con la que obtiene el estado de un puerto concreto del GPIO, realizando un acceso a la dirección del registro IN del puerto GPIO donde se encuentre mapeado el pin sobre el que quiere realizarse la consulta. Con la macro PORT se obtiene el índice del GPIO, 0 o 1, haciendo un shift de 5 bits hacia la derecha del argumento que recibe la función *gpio_in*, y que se compone de las dos partes siguientes: el quinto bit más significativo indica el puerto del GPIO y, los cuatro bits menos significativos indican el índice del pin al que se encuentra conectado el dispositivo. Este último valor es empleado en la función GET_BIT para realizar un offset hacia la derecha sobre el valor leído del registro de estado, de forma que el LSB en dicho valor sea el bit que indica el estado del dispositivo. Una vez hecho el offset, la macro limpia el resto de bits más significativos que el LSB aplicando una máscara con el valor 0x1, realizando la operación AND bit a bit de la máscara con el valor leído del registro. El resultado es entonces retornado por la función *gpio_in*.

```
#define GET_BIT(reg, n) (((reg) >> (n)) & 0x1)
#define PORT(x) ((x)>>5)
#define PIN(x) ((x)&0x1f)

inline unsigned gpio_in(unsigned pin) {
    return GET_BIT(GPIO[PORT(pin)]->IN, PIN(pin));
}
```

Figura 23: Función y macros empleadas en la lectura del estado de un pin del GPIO

6.4. Manejador del altavoz

Una de las posibles aplicaciones de este dispositivo es la creación de melodías sencillas. Es por ello por lo que se ha dotado a la librería de capacidad para reproducir tonos de distintas frecuencias durante periodos de duración variable, con los que los alumnos podrán crear sus propias melodías.

Las oscilaciones del elemento pulsátil del altavoz, el cual está basado en un zumbador piezoeléctrico, son generadas mediante la aplicación de tensiones de alimentación alternas sobre el pin del GPIO al que se encuentra conectado, es decir, periodos en los que se aplica un valor de tensión positivo, seguido de un valor de tensión nulo. Como puede verse en el pseudocódigo de la Figura 26, entre cada pulso se espera un tiempo igual a la mitad del periodo del tono que vaya a reproducirse, creándose así cada dos pulsos un ciclo completo del tono. Estos periodos resultan del cálculo de la inversa de las frecuencias de cada tono [15], son almacenados en un array (*periodo_us* en la Figura 25), y generan las notas desde el Do 4, a 261.63 Hz, con un periodo de $1/261.63 \approx 3822$ µs, hasta el Do 6, a 1046.5 Hz, con $1/1046.5 \approx 955$ µs de periodo. Obtenemos los periodos en microsegundos puesto que es la unidad que recibe la función *delay_loop*, con la que se generan el espaciado entre los pulsos ascendentes y descendentes en el zumbador.

Para cambiar el valor de la salida en un pin del GPIO se invoca la función *gpio_out* de Microbian, con la que se escribe el bit correspondiente en los registros OUTSET o OUTCLR, según se quiera establecer, respectivamente, un valor de tensión positivo o nulo en el pin. No obstante, Microbian no contaba con soporte para el zumbador, por lo que fue preciso definir su identificador. Este identificador (0b0), se corresponde con el pin 0 del puerto GPIO 0, y se incorpora a Microbian mediante la definición en el fichero *hardware.h* detallada en la Figura 24.

```
#define BUZZER DEVPIN(0,0)
```

Figura 24: Definición del identificador del zumbador en *hardware.h*

```
unsigned int periodo_us[] = {3822, 3405, 3033, 2863, 2551, 2272, 2024, 1911,1702,1516,
                             1431, 1275, 1136, 1012, 955};
```

Figura 25: Definición del array de periodos en el driver del zumbador

Al igual que se hizo en el driver de los botones, en el del zumbador se definen también una serie de enumerados con los que los alumnos podrán referirse a los distintos tonos que podrán generar con el dispositivo. Estos enumerados constituyen el tipo de dato *nota_t*, cuya definición se detalla en la Figura 27. Aunque no lleguen a comprender en un primer momento las cualidades de un tipo enumerado, se espera que este sistema les resulte cuanto menos intuitivo. Cuando quieran reproducir una nota con el zumbador, simplemente deberán llamar a la función *buzzer_reproduce_nota* indicando el tono y su duración.

```
FUNCIÓN buzzer_reproduce_nota (nota, duración):
INICIO
        espera_entre_pulsos ← nota.periodo / 2
        ciclos ← duración / nota.periodo
        REPETIR ciclos VECES
                Zumbador.enciende
                Espera espera_entre_pulsos microsegundos
                Zumbador.apaga
                Espera espera_entre_pulsos microsegundos
FIN
```

Figura 26: Pseudocódigo de la función *buzzer_reproduce_nota*

```
typedef enum {DO_4, RE_4, MI_4, FA_4, SOL_4, LA_4, SI_4, DO_5, RE_5,
              MI_5, FA_5, SOL_5, LA_5, SI_5, DO_6} nota_t;
```

Figura 27: Definición del tipo de dato *nota_t*

6.5. Manejador de la matriz de LEDs

Se trata sin duda del dispositivo que más juego ha dado durante el desarrollo del proyecto. Inicialmente se pensó en desarrollar únicamente las funciones básicas para el manejo de LEDs individuales, pero dado el amplio abanico de posibilidades que ofrece el display, se decidió implementar un conjunto más amplio de funciones, cuyo desarrollo también podría ser propuesto a modo de práctica.

El driver del display que disponible en Microbian cuenta con una función de inicialización (*display_init*) la cual crea un proceso encargado de realizar, de manera periódica, el refresco de la imagen mostrada en el display, a partir de los valores almacenados en una variable del tipo *image* que contiene el estado de los LEDs del display a cada momento.

6.5.1. Configuración de la intensidad en la matriz de LEDs

Es posible, a través de la función *gpio_drive* de Microbian, establecer la tensión de alimentación en los pines de las interfaces GPIO, alterando el valor de los bits 8, 9 y 10 del registro PINCNF correspondiente al pin que quiera configurarse [16].

La función de la librería *ubit*, *display_cambia_intensidad*, recibe como parámetro un enumerado cuya definición se proporciona en la Figura 28. A través de este tipo de dato, los alumnos podrán indicar el nivel de intensidad que se desee establecer en el display, dentro de tres valores posibles. Microbian ya posee constantes para indicar el nivel de brillo de los LEDs, pero sus nombres no son descriptivos del nivel de intensidad a no ser que se consulte la especificación del microcontrolador. Es por ello por lo que se decidió proporcionar un conjunto de enumerados con los que los alumnos pudieran, de una forma sencilla e intuitiva, indicar a la placa la intensidad con la que quieran que luzcan sus LEDs.

```
typedef enum {INT_BAJA, INT_MEDIA, INT_ALTA} intensidad_t;
```

Figura 28: Definición del tipo de dato *intensidad_t*

6.5.2. Encendido y apagado de LEDs individuales

Se proporcionan dos funciones básicas con las que operar a nivel de LEDs individuales, encendiéndolos o apagándolos, y sus respectivas implementaciones se detallan en la Figura 29.

En *display_enciende_led* se llama a las funciones de Microbian *image_set* y *display_show*. La primera se emplea para generar sobre una imagen (del tipo *image*) el conjunto de señales de control que hagan que se encienda un LED en una posición determinada cuando la tarea de refresco muestre la fila donde se encuentra dicho LED. La segunda simplemente copia el conjunto de valores de una imagen sobre la variable del driver que alberga el estado actual de la imagen que se muestra en el display.

El driver del display no contaba con una función para apagar un LED en una imagen, por lo que fue necesario implementar la función *image_clear_led*, que genera sobre la imagen el conjunto de señales que hace que un LED en una posición determinada dentro de la matriz se muestre apagado. Podría pensarse que en *display_apaga_led* con alterar directamente los contenidos de los registros de control para apagar un LED concreto sería suficiente, pero esto no tendría sentido, puesto que ese LED va a seguir "encendido" en la imagen del driver, con lo que el refresco volverá a encenderlo.

```
int
display_enciende_LED(int x, int y)
{
    if (x < 0 || x >= DISPLAY_DIM || y < 0 || y >= DISPLAY_DIM) return -1;

    image_set(x, y, imagen_actual_microbian);
    display_show(imagen_actual_microbian);

    return 0;
}

int
display_apaga_LED(int x, int y)
{
    if (x < 0 || x >= DISPLAY_DIM || y < 0 || y >= DISPLAY_DIM) return -1;

    image_clear_led(x, y, imagen_actual_microbian);
    display_show(imagen_actual_microbian);

    return 0;
}
```

Figura 29: Funciones para el encendido y apagado de LEDs individuales

6.5.3. Mostrando imágenes en el display

Cuando quiere mostrarse una imagen en el display haciendo uso única y exclusivamente de la funcionalidad proporcionada por el driver, esta debe construirse con el tipo de dato *image*, y la macro IMAGE, cuya implementación se proporciona en la Figura 30. Esta macro recibe un listado de enteros, 0 o 1, y los convierte en el conjunto de señales de control que configurarán los pines de las interfaces GPIO 0 y 1 de tal forma que se muestre la imagen deseada. La tarea de refresco del display recorre las cinco filas de la imagen de Microbian que alberga el estado del display a cada momento y, por cada fila, obtiene los valores de las señales de control de cada interfaz GPIO y escribe esos valores en los registros OUTSET de dichas interfaces. Este registro controla la configuración del voltaje de salida de los pines, resultando la escritura de un bit con valor 1 en la aplicación de un voltaje positivo en el pin y, un 0, en la conservación del voltaje de salida previo a la escritura del valor [17]. Las interfaces cuentan además con un registro OUTCLR cuyo propósito es el contrario del registro OUTSET, es decir, el apagado de los pines del GPIO cuando se escribe algún bit con valor 1 [18].

```
const unsigned imagen_ejemplo[] =
    IMAGE(0,1,0,1,0,
          1,0,1,0,1,
          0,1,0,1,0,
          1,0,1,0,1,
          0,1,0,1,0);
```

Figura 30: Ejemplo de creación de una imagen de Microbian

El tipo de dato *image* que provee Microbian para la creación de imágenes, aunque facilita sustancialmente la creación de imágenes que puedan ser mostradas por el driver del display, desde el punto de vista de la docencia de la programación en la asignatura de Introducción al Software no resulta de mucha utilidad o, mejor dicho, no da tanto juego como podría dar, por ejemplo, un array bidimensional y es que, con objeto de que los alumnos practiquen con este tipo de estructuras, la librería *ubit* les proporciona un tipo de dato *imagen_t*, con el que podrán describir sus imágenes como matrices cuyos elementos serán ceros y unos.

Fue necesario implementar una función para mostrar imágenes del tipo *imagen_t* en el display. La nueva función (*display_muestra_imagen*), cuyo pseudocódigo se detalla en la Figura 31, recibe como parámetro un dato del tipo *imagen_t* y recorre la matriz trasladando la información contenida en ella a una imagen de Microbian (del tipo *image*) mediante la función *image_set*. Esta última imagen, que ya contiene las señales de control de los LEDs, será mostrada posteriormente con la función *display_show*. Los alumnos podrán también mostrar secuencias de imágenes en el display y crear sus propias animaciones gracias a la función *display_muestra_secuencia*, cuyo pseudocódigo de detalla en la Figura 32.

```
FUNCIÓN display_muestra_imagen(imagen):
INICIO
       Resultado ← imagen vacía
       POR CADA píxel EN imagen:
              SI píxel encendido ENTONCES
                     Resultado(mismas coordenadas) ← encendido
       Imagen driver ← Resultado
FIN
```

Figura 31: Pseudocódigo de la función *display_muestra_imagen*

```
FUNCIÓN display_muestra_secuencia(secuencia, delay):
INICIO
       POR CADA imagen EN secuencia:
              display_muestra_imagen(imagen)
              espera(delay)
FIN
```

Figura 32: Pseudocódigo de la función *display_muestra_secuencia*

6.5.4. Mostrando texto

Para completar el abanico de herramientas con que poder manejar, experimentar y crear con el display, se proporciona una función para mostrar texto con una animación de deslizamiento hacia la izquierda. Esta función recibe, haciendo una excepción respecto del criterio de no emplear paso de argumentos por referencia, un puntero a una cadena de caracteres y un enumerado indicando la velocidad a la que el texto deberá deslizarse por el display. La velocidad del texto se define, una vez más, como un tipo de dato enumerado, con tres valores posibles, puesto que se consideró que sería más visual o intuitivo para los alumnos el indicar la velocidad por medio de una "palabra reservada" que mediante un valor numérico. Se detalla en la Figura 33 la definición de este nuevo tipo de dato.

```
typedef enum {LENTO, MEDIO, RAPIDO} velocidad_texto_t;
```

Figura 33: Definición del tipo de dato *velocidad_texto_t*

No existe forma alguna de convertir, en tiempo de ejecución, un carácter ASCII en un array bidimensional con el conjunto de ceros y unos que hagan que se muestre dicho carácter en el display. Es por ello por lo que se hace necesario guardar de algún modo las codificaciones de los posibles caracteres que los alumnos puedan querer mostrar. En un primer momento podría pensarse en un array de datos del tipo *imagen_t* que guarde directamente todas las imágenes de los caracteres, pero debe tenerse en cuenta que, como mínimo, serían necesarias 36 imágenes para representar en el display los números del 0 al 9 y las 26 letras del alfabeto latino estándar. Esta aproximación daría como resultado un array de 5 $(dim.\,display) \times 5 \times 4\ (bytes/elem.) \times 36 = 3600\ Bytes$, por lo que no parece la mejor forma de construir una librería de caracteres. Una opción más viable, en términos de almacenamiento al menos, sería el guardar la imagen de cada carácter en un formato más compacto como, por ejemplo, en un string.

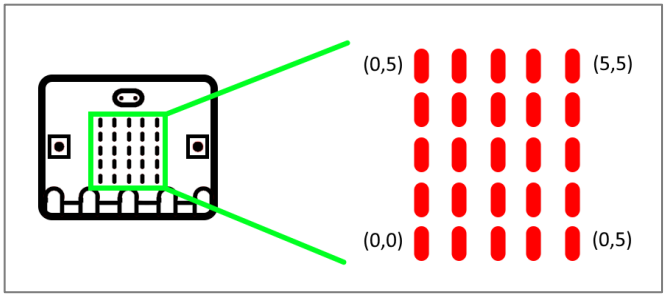

Figura 34: Localización de los LEDs en la matriz mediante un sistema de coordenadas cartesianas

Esta última es la forma en que la librería *Ubit* mantiene internamente una librería de sprites (*sprites.h*), dentro de la cual se define un array de 63 cadenas de 26 caracteres cada una (25 más el terminador nulo), donde cada uno indica el estado de un led en la matriz: el primer carácter

indica el estado del LED en la posición (0,0), el segundo el estado del LED en la (0,1), el sexto estaría en la (1,0), etc. (puede consultarse el sistema de localización de los LEDs en la placa mediante coordenadas cartesianas en la Figura 34). De este modo se reduce el tamaño de la colección básica de sprites a $(5\ (dim.\ display) \times 5\ (dim.\ display) + 1\ (terminador\ nulo) \times 36 = 936\ Bytes$, aproximadamente un cuarto del tamaño de la primera opción. No obstante, y para dotar de mayor flexibilidad a la librería, se han generado las codificaciones de los caracteres ASCII del 33 al 95.

Para generar las codificaciones de los caracteres se desarrolló el programa auxiliar *png2string* (cuyo flujo de ejecución se detalla en la Figura 35) con el que poder convertir ficheros de imagen en formato *.png* en sus equivalentes "codificaciones binarias", con objeto de agilizar su obtención, tal y como puede verse en el diagrama de la Figura 36. El programa recibe dos argumentos por la línea de comandos: un listado con los nombres de los ficheros de imagen a procesar, y el nombre del fichero donde se desea depositar el resultado de la conversión y, en caso de no indicar el segundo, se le da un nombre por defecto. Además, la herramienta genera, como puede verse en la Figura 38, la conversión en un formato adecuado para su inclusión en un array de cadenas de caracteres, de forma que para generar la colección de sprites tan solo fue necesario copiar y pegar la salida en el fichero de cabecera. Además, esto permitirá a los alumnos diseñar sus propias animaciones en la herramienta de edición de imágenes que prefieran, fotograma a fotograma, convertir esos diseños en sus strings equivalentes y, por medio de la función *display_muestra_sprite*, convertirlos en su equivalente imagen de *ubit* y mostrarlos en el display de la placa.

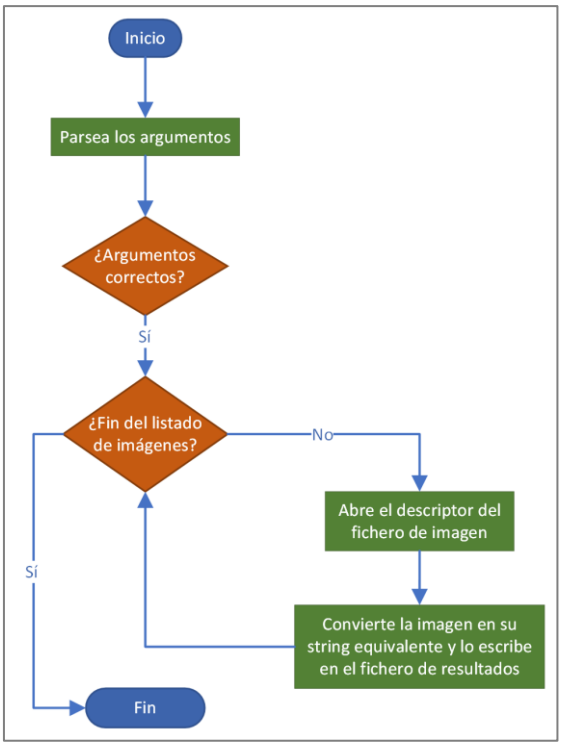

Figura 35: Flujo de ejecución de la herramienta *png2string*

También se proporciona una función *display_char2codi* con la que obtener la codificación un carácter a partir de su valor decimal, accediendo con dicho valor menos 33 al array de caracteres en *sprites.h*. Se resta el valor 33 al valor decimal del carácter puesto que es el 33 el primer carácter (exclamación de cierre, ASCII 33) en el array. Esta función es además la empleada en *display_muestra_texto* para obtener las codificaciones de cada carácter en la cadena.

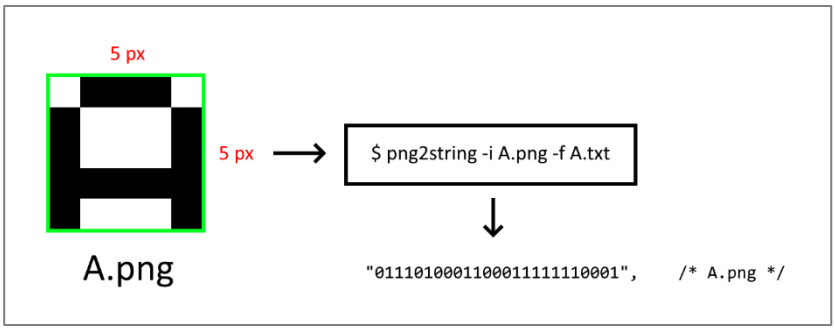

Figura 36: Proceso de conversión de una imagen en la secuencia de ceros y unos que hará que se muestre en el display

```
FUNCIÓN display_muestra_texto(texto, velocidad):
INICIO
        codificación_carácter_1 ← codificación del primer carácter del texto
        codificación_carácter_2 ← vacío

        PARA CADA carácter EN texto:
                codificación_carácter_2 ← codificación del carácter
                Obtener imágenes a partir de las codificaciones

                REPETIR HASTA que los dos caracteres se hayan mostrado
                        Obtener composición de los dos caracteres
                        Mostrar imagen compuesta
                        Esperar (base / velocidad) milisegundos

                codificación_carácter_1 ← codificación del siguiente carácter
FIN
```

Figura 37: Pseudocódigo de la función *display_muestra_texto*

```
[ [redacted] ] : [redacted]/Micro-bit-C-API/herramientas/sprites > (desarrollo)
$ ./png2string -i $(ls | grep .png) -f ASCII.txt
[*] Procesando "33_exclamacion_cierre.png"
[*] Procesando "34_comillas.png"
[*] Procesando "35_almohadilla.png"
[*] Procesando "36_dolar.png"
[*] Procesando "37_porcentaje.png"
[*] Procesando "38_ampersand.png"
[*] Procesando "39_comilla_simple.png"
[*] Procesando "40_parentesis_apertura.png"
    • • •
[*] Procesando "80_P.png"
[*] Procesando "81_Q.png"
[*] Procesando "82_R.png"
[*] Procesando "83_S.png"
[*] Procesando "84_T.png"
[*] Procesando "85_U.png"
[*] Procesando "86_V.png"
[*] Procesando "87_W.png"
[*] Procesando "88_X.png"
[*] Procesando "89_Y.png"
[*] Procesando "90_Z.png"
[*] Procesando "91_corchete_apertura.png"
[*] Procesando "92_barra_invertida.png"
[*] Procesando "93_corchete_cierre.png"
[*] Procesando "94_acento_circunflejo.png"
[*] Procesando "95_barra_baja.png"
[*] Conversión guardada en "ASCII.txt"
```

Figura 38: Proceso de obtención de las cadenas de caracteres que conforman la librería de sprites

Finalmente, la función *display_muestra_texto*, cuyo pseudocódigo puede verse en la Figura 37, es la encargada de realizar la conversión de la cadena de caracteres que le pase el usuario, en la secuencia de imágenes de *ubit* que haga que esa cadena se muestre en el display de 5x5 LEDs de la placa, con una animación de deslizamiento hacia la izquierda.

En cada iteración del bucle externo se pasa por todos los caracteres de la cadena, y se trabaja siempre con el carácter que corresponda a esa iteración y con el siguiente, de forma que en el bucle interno se vaya generando una composición de ambos caracteres, con partes de uno y partes del otro, con lo que se irá generando la animación hasta que los dos hayan sido mostrados por completo.

6.6. Manejador del sensor de temperatura

El microcontrolador cuenta con un sensor de temperatura [19], no ambiental si no del propio micro. No obstante, este dato puede ser una aproximación más o menos precisa del valor real de la temperatura ambiental.

El procedimiento para obtener una lectura del sensor es realmente sencillo, y se detalla en el pseudocódigo de la Figura 39.

El dato depositado por el sensor en el registro TEMP cuando se activa en EVENTS_DATARDY la señal que indica que hay un dato disponible, no es el dato de temperatura como tal, sino que este viene expresado en número de incrementos de 0.25 °C, por lo que el dato real será el valor leído del registro multiplicado por esa cifra.

```
FUNCIÓN temometro_lectura():
INICIO
       Escribe 1 en el registro TASKS_START
       Espera trigger en el registro EVENTS_DATARDY
       Lee registro TEMP
       Retorna el producto del valor leído por 0.25
FIN
```

Figura 39: Lectura del dato de temperatura

6.7. Descarga e instalación de la librería

Se ha procurado facilitar la distribución tanto de la librería como de la utilería desarrollada por medio de un paquete de software en formato *.deb* cuya instalación permitirá a los alumnos hacer uso de todas las herramientas desarrolladas sin tener que realizar configuraciones adicionales en sus sistemas operativos. Este paquete se encuentra alojado en el repositorio del proyecto, el cual se ha hecho público, con objeto de que cualquiera que vea a bien realizar modificaciones sobre el software desarrollado para adaptarlo a sus necesidades o mejorarlo, o simplemente desee hacer uso de la librería y las herramientas, pueda hacerlo libremente.

El paquete recibe un nombre conforme al nombramiento estándar empleado en los paquetes de Debian. Se detalla en la Figura 41 la estructura del paquete, el cual instalará la librería (*libubit.a*), los ficheros de cabecera, tanto de la librería Ubit como del sistema operativo, el linker script de Microbian (*nRF52833.ld*) y las herramientas *ubit-gcc*, *ubit-load* y *png2string*. Las herramientas de compilación y carga fueron compiladas, valga la redundancia, mediante la utilidad *shc*, de la cual se proporciona un ejemplo de uso en la Figura 40. De esta forma, en el paquete software se distribuyen los ejecutables resultantes, y no los scripts.

```
[ [redacted] ] : [redacted]/Micro-bit-C-API/herramientas > (desarrollo)
$ shc -vrf ubit-gcc.sh -o ubit-gcc
```

Figura 40: Obtención de ejecutables con la herramienta *shc*

La instalación se realiza sobre el directorio /usr/local/, destinado en los sistemas de ficheros de la gran mayoría de distribuciones Linux basadas en Debian a la instalación de software por parte de los administradores. En /usr/bin/ se depositan los ejecutables de las herramientas, en /usr/include/ los ficheros de cabecera y, en /usr/share/ubit/ el linker script y la librería Ubit.

El fichero *DEBIAN/control* alberga la información básica sobre el paquete (nombre, versión, descripción, etc.), el cual ha sido creado por medio de la herramienta *dpkg-deb* y puede ser instalado de una manera sencilla mediante el comando *dpkg* con la opción -i, como puede verse en la Figura 42.

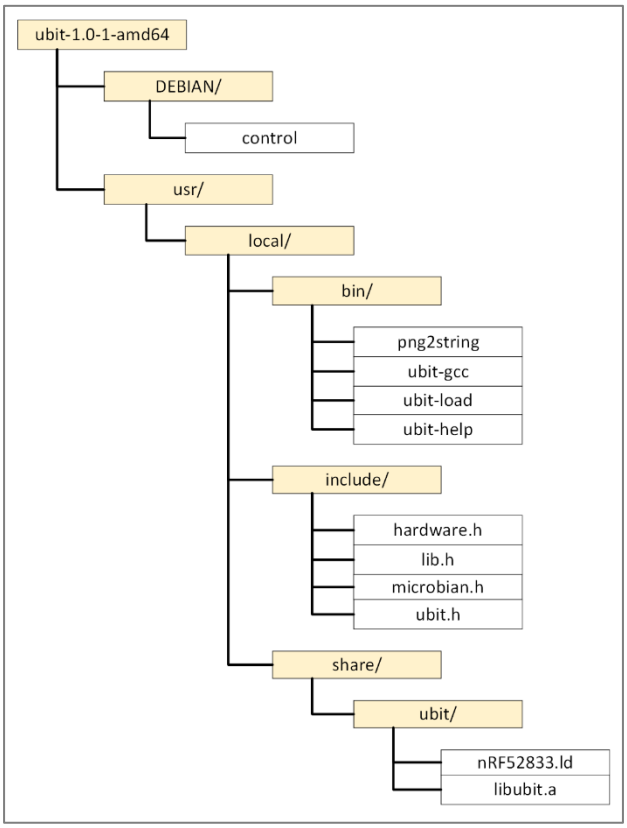

Figura 41: Distribución interna del paquete *.deb*

```
$ dpkg -i ubit-1.0-1-amd64.deb
```

Figura 42: Instalación del paquete

El paquete se construye con el comando de la Figura 43, indicando a la herramienta el flag *--root-owner-group*, con el que se configuran todos los ficheros del paquete para pertenecer al usuario y grupo *root*.

```
$ dpkg-deb --root-owner-group --build ubit-1.0-1-amd64/
```

Figura 43: Creación del paquete

6.8. Documentación

Durante todo el desarrollo del proyecto se ha ido generando la documentación asociada al mismo o, mejor dicho, se fue preparando por medio de los comentarios y etiquetas con que posteriormente la herramienta Doxygen podría generarla. Además, y dado que el estilo por

defecto del sitio web que se genera se encuentra algo desactualizado, se empleó la misma hoja de estilos de la asignatura de Introducción al Software, así como el fichero de configuración empleado en la generación de la documentación de dicha asignatura.

Para generar la documentación, tan solo es preciso dirigirse a la raíz del proyecto, en el caso que nos ocupa, la raíz de la librería ubit, es decir, el directorio, que alberga los ficheros de la librería de alto nivel, y ejecutar desde ahí el comando de la Figura 44. Como resultado, se generará en el directorio de salida indicado en el fichero de configuración (Doxyfile) la documentación en el formato configurado en dicho fichero: por un lado un sitio web del cual se proporciona una muestra en la Figura 45 y, por otro lado, toda la documentación en formato PDF.

```
$ doxygen <ruta al Doxyfile>
```

Figura 44: Creación de la documentación por medio de la herramienta Doxygen

Figura 45: Sitio web generado por Doxygen

6.9. Pruebas

Se han generado múltiples ficheros de código fuente para comprobar el correcto funcionamiento de todas y cada una de las funciones desarrolladas para la librería Ubit. Todos estos códigos de prueba se encuentran disponibles en el repositorio del proyecto, y son los que se muestran en la Figura 46.

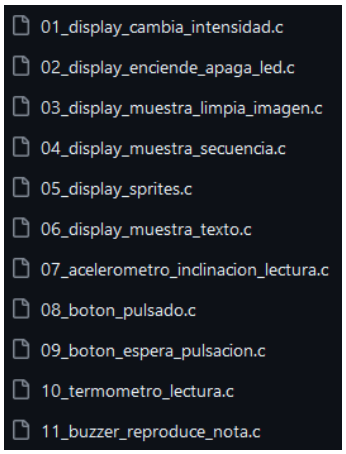

Figura 46: Códigos de prueba de las funciones desarrolladas

6.9.1. Pruebas del manejador de la matriz de LEDs

Se comenzó por probar la función para realizar el cambio de intensidad en el display, siguiendo el algoritmo descrito en la Figura 47, es decir, realizando un total de cuatro cambios de intensidad graduales, de más baja a más alta, y un último cambio de intensidad con un valor no válido, con objeto de poner a prueba la comprobación de parámetros en la función *display_cambia_intensidad*.

```
TEST display_cambia_intensidad:
INICIO
       PARA CADA nivel de intensidad HACER:
              Cambia el nivel de intensidad
              Comprueba el valor de retorno de la función
              Espera 1 segundo
       FIN PARA
FIN
```

Figura 47: Pseudocódigo de la prueba de la función *display_cambia_intensidad*

También se ha probado el correcto funcionamiento de las funciones de encendido y apagado de LEDs individuales, por medio de un programa cuyo algoritmo se proporciona en la Figura 48. Esta prueba realiza el encendido progresivo de todos los LEDs de la matriz, comenzando por el situado en la posición (0,0), y siguiendo con esa fila hasta llegar al final. En ese momento, enciende el último LED de la fila inmediatamente superior y continúa hasta el comienzo de esa fila, de tal forma que se vaya generando una animación de zig-zag hasta llegar al LED en la posición (5,5). A partir de ahí se genera la misma animación, pero en el sentido contrario, apagando progresivamente todos los LEDs de la matriz.

```
TEST display_encience_apaga_LED:
INICIO
       PARA CADA fila HACER:
              PARA CADA columna HACER:
                     SI fila es par ENTONCES
                            Encender LED en posición (columna, fila)
                     SI NO ENTONCES
                            Encender LED en posición (dim. display – 1
                                                   – columna, fila)
              FIN PARA
       FIN PARA
       PARA CADA fila HACER:
              PARA CADA columna HACER:
                     SI fila es par ENTONCES
                            Apagar LED en posición (columna, fila)
                     SI NO ENTONCES
                            Apagar LED en posición (dim. display – 1 – columna,
                                                   fila)
              FIN PARA
       FIN PARA
```

Figura 48: Pseudocódigo de la prueba de las funciones *display_enciende_led* y *display_apaga_led*

Para probar que las imágenes se muestran correctamente en el display, se desarrolló una prueba que muestra dos imágenes distintas una tras de otra y, después de estas, apaga todos los LEDs de la matriz. De esta manera queda demostrado el correcto funcionamiento de las funciones *display_muestra_imagen* y *display_limpia*. Asimismo, se probó el funcionamiento de *display_muestra_secuencia* mostrando en el display un conjunto de cinco imágenes cada una de las cuales enciende una fila distinta en la matriz.

Por último, se pusieron a prueba las funciones relacionadas con el uso de los sprites de Ubit, comprobando que todos y cada uno de los caracteres soportados por la librería se mostraban correctamente. Además, al final de esta última prueba, se muestra un texto en el display el cual contiene todos los caracteres de la librería.

6.9.2. Pruebas del manejador del sensor de movimiento

Se desarrolló una única prueba en la que continuamente se envían las lecturas del sensor de aceleración a través de la interfaz serie, así como los valores de inclinación calculados por las funciones *acelerometro_inclinacion_eje_x/y* de forma que, mediante un transportador de ángulos, pudiera comprobarse la exactitud de dichos valores.

6.9.3. Pruebas del manejador de los botones

Se han desarrollado dos pruebas. En la primera, se muestra en el display el carácter distintivo del botón cuya pulsación haya sido detectada por medio de la función *boton_pulsado*. Por ejemplo, cuando se presiona el botón de pulsación izquierdo, el cual aparece marcado en la placa con la letra 'A', se muestra esa letra en el display, o cuando se pulsa sobre el botón táctil con el logo de Micro:bit, se muestra la letra 'M'.

En la segunda prueba se genera, mediante la función *boton_espera_pulsación*, una secuencia de pulsaciones en un orden concreto, mostrando el carácter identificativo de cada botón cuando estos son pulsados en el orden marcado por la lógica del programa.

6.9.4. Pruebas del manejador del sensor de temperatura y el manejador del zumbador

Para comprobar el correcto funcionamiento del primero, simplemente se envían constantemente las lecturas obtenidas por el sensor a través de la interfaz serie, de forma que estas puedan ser comparadas con las lecturas de un termómetro situado en el mismo entorno que la placa y, sobre el segundo, la prueba desarrollada genera una escala con todos los tonos soportados por el manejador del zumbador, y con distintas duraciones.

7. Demostradores

Se han desarrollado tres programas con los cuales demostrar las capacidades tanto de la placa Micro:bit como de la librería desarrollada. Estos desarrollos podrían a su vez ser empleados en las prácticas de la asignatura Introducción al Software, bien como demostradores de los proyectos que los alumnos podrían llegar a realizar, o bien como prácticas reales donde poder aplicar los conocimientos sobre programación en el lenguaje C impartidos en las sesiones de teoría.

Los programas, en el orden en que serán presentados en los apartados siguientes, han sido pensados para poseer una complejidad incremental.

7.1. Herramienta de nivelación

Se trata, como su nombre indica, de una simulación de un nivel de burbuja, con el que poder obtener una aproximación del paralelismo de la placa respecto del plano perpendicular a la recta que la une virtualmente con el centro de masas del planeta.

Observando el diagrama del flujo de ejecución del programa en la Figura 49, puede obtenerse una idea de su sencillez. Se trata del más sencillo de los tres programas desarrollados, y en él se hace uso de los siguientes dispositivos:

- El acelerómetro, para obtener el valor de la inclinación en los ejes X e Y
- El display, en el que se emula el comportamiento de la burbuja de un nivel

Los alumnos pondrán en práctica el uso de los tipos básicos de datos, así como de las sentencias de control *if* y *while*.

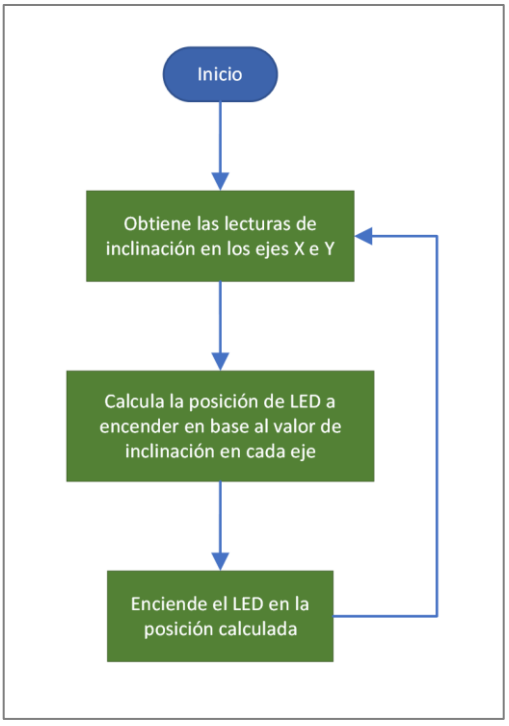

Figura 49: Flujo de ejecución de la demo *nivel.c*

7.2. “Conway’s Game Of Life”

Haciendo uso del display LED se ha realizado una implementación del “Juego de la Vida”, el cual es un autómata celular con el que se simula la evolución de una población de células con capacidad para autorreplicarse siguiendo el algoritmo descrito en la Figura 51, en el cual se aplican tres reglas básicas [20]:

1. Si una célula vive y tiene 2 o 3 vecinos, pasará a la siguiente generación
2. Si una célula viva tiene menos de 2 o más de 3 vecinos muere, por aislamiento o sobrepoblación respectivamente
3. Una célula nace si y sólo si tiene exactamente 3 vecinos

Ejemplos de la aplicación de estas reglas pueden encontrarse a continuación en la Figura 50.

En este proyecto los alumnos podrán poner en práctica las operaciones con enteros y arrays bidimensionales, así como el uso de las sentencias de control *if*, *while* y *for*.

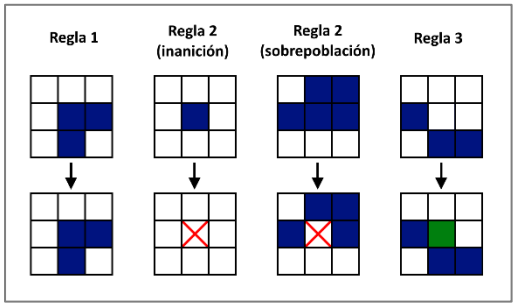

Figura 50: Ejemplos de evolución en el Juego de la Vida

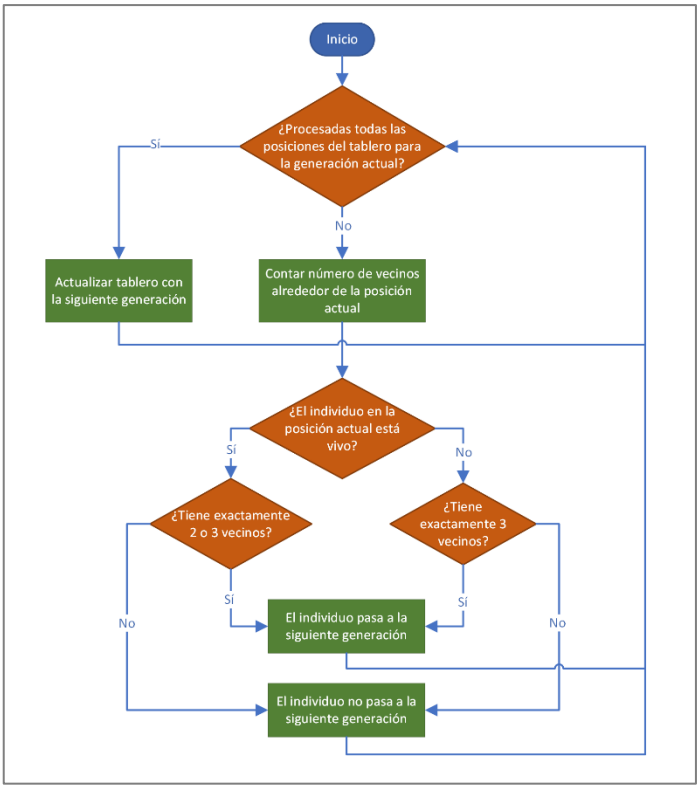

Figura 51: Diagrama de flujo describiendo el funcionamiento del Juego de la Vida

7.3. Laberinto

Se trata de una implementación del clásico juego del laberinto, la cual hace uso del acelerómetro para interpretar los gestos del usuario y hacer que el elemento móvil se vaya desplazando a través de distintos niveles que son mostrados en la matriz de LEDs.

La idea con este proyecto sería darle un enfoque incremental, proponiendo a los alumnos en un primer momento, que implementasen la funcionalidad que les permita hacer que la entidad que maneja el jugador se desplace a través del display en base a los movimientos del usuario. La entidad móvil que maneja el jugador (marcada en color rojo en la Figura 52) deberá desplazarse en el sentido y dirección que marque la inclinación de la placa.

En esta parte, los alumnos harían uso del acelerómetro para detectar la inclinación de la placa, y la matriz de LEDs para mostrar los movimientos del jugador.

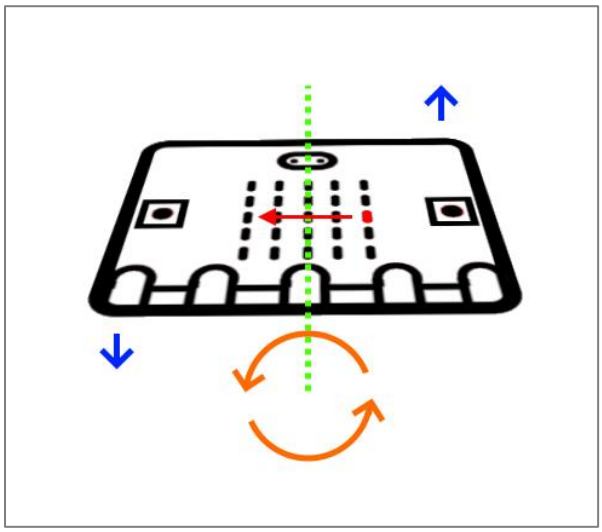

Figura 52: Representación del comportamiento esperado cuando se inclina la placa en sentido antihorario respecto del eje Y

Una vez el alumno sea capaz de generar los movimientos de la entidad móvil a través del display, le podría ser encomendada la tarea de diseñar un nivel y programar el comportamiento de la bola cuando esta entra en contacto con las paredes del laberinto, es decir, que debería implementar un pequeño motor de colisiones con el que simular el comportamiento que tendría la bola en el mundo físico al desplazarse por el laberinto. Se proporciona en la Figura 53 el pseudocódigo de una posible implementación del motor de colisiones.

Por último, podría implementarse un sistema de niveles, de tal forma que los laberintos tuvieran un punto de inicio y una meta y, al llegar al meta, se pasara al siguiente nivel, hasta llegar al último. En ese momento, y con el objetivo de que los alumnos empleen tantos dispositivos hardware como sea posible, podría reproducirse una melodía en el zumbador, y mostrar un mensaje de victoria en el display.

En la Figura 54 se muestran los tres niveles de que consta la implementación realizada a modo de demostración, marcando en color verde y azul respectivamente, los puntos de comienzo y final de cada uno.

```
FUNCIÓN procesa_colisión():
INICIO
       SI placa inclinada hacia la izda. Y
          jugador no tocando el borde izdo. del display Y
          jugador no tiene un borde del laberinto a la izda.
       ENTONCES
              mueve jugador hacia la izda.
       SI NO SI placa inclinada hacia la dcha. Y
          jugador no tocando el borde dcho. del display Y
          jugador no tiene un borde del laberinto a la dcha.
       ENTONCES
              mueve jugador hacia la dcha.

       SI placa inclinada hacia delante Y
          jugador no tocando el borde superior del display Y
          jugador no tiene un borde del laberinto encima
       ENTONCES
              mueve jugador hacia arriba
       SI NO SI placa inclinada hacia abajo Y
          jugador no tocando el borde inferior del display Y
          jugador no tiene un borde del laberinto debajo
       ENTONCES
              mueve jugador hacia abajo
FIN
```

Figura 53: Pseudocódigo de una posible implementación del motor de colisiones

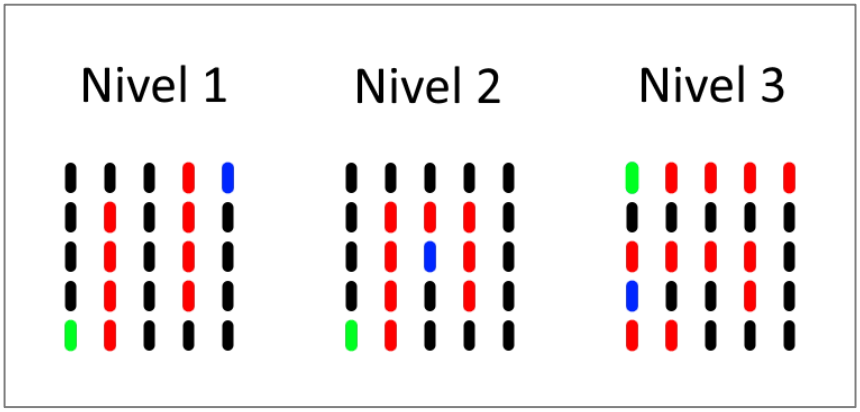

Figura 54: Niveles en la implementación desarrollada a modo de demostración

8. Conclusiones y trabajo futuro

El objetivo primero del proyecto era la creación de una librería de alto nivel con la que los alumnos pudieran aplicar los conocimientos adquiridos en la asignatura de Introducción al Software sobre una plataforma hardware que, por sus características, pudiera hacer de las sesiones prácticas una experiencia mucho más motivadora.

La librería desarrollada dota a los alumnos de capacidad para interactuar con una variedad de dispositivos lo suficientemente amplia como para introducirles de lleno tanto en la programación en lenguaje C, como en la programación a alto nivel de sistemas embebidos sencillos con los que podrán reforzar la asimilación de los conceptos de la programación en general, así como los aspectos concretos del lenguaje en que se encuentra enfocada la librería. Concretamente, se han implementado manejadores para los botones de la placa, el zumbador, el acelerómetro y el sensor de temperatura, los cuales no contaban originalmente con un soporte directo por parte del sistema operativo Microbian, si no que este proporcionaba el conjunto de llamadas que permitían interactuar a bajo nivel con las interfaces a las que se encontraban conectados estos dispositivos. Asimismo, se ha extendido la funcionalidad del manejador original de la matriz de LEDs y se ha simplificado su uso.

Adicionalmente, y en lo que respecta a las herramientas de apoyo desarrolladas, se ha procurado mantener una estructura similar al proceso de compilación nativa que hasta ahora se ha venido utilizando en las asignaturas introductorias a la programación. Para ello, se han desarrollado las herramientas *ubit-gcc* y *ubit-load*, que simplifican el proceso de compilación y de carga de ejecutables, respectivamente.

Micro:bit es el candidato ideal para ser integrado en la docencia de las asignaturas de programación en cualquiera de los niveles educativos del sistema español por dos motivos principales: por un lado su bajo coste y, por otro lado, la gran variedad de dispositivos hardware con los que poder interactuar.

Como trabajo futuro, podría sugerirse la implementación de llamadas de alto nivel para las comunicaciones por radio a través de las llamadas de Microbian o, incluso, la implementación de un driver a nivel del sistema operativo para establecer comunicaciones a través de Bluetooth BLE. Además, quedaría pendiente la implementación de las funciones de alto nivel para el manejo del magnetómetro integrado en el sensor de movimiento.

Bibliografía

[1] BBC. (6 de Julio de 2015). *Legacy - BBC Media Centre*. Obtenido de BBC Media Centre: https://www.bbc.co.uk/mediacentre/mediapacks/microbit/legacy/

[2] Spivey, M. (18 de Diciembre de 2022). *The Microbian Project - Digital Systems*. Obtenido de Spivey's Corner: https://spivey.oriel.ox.ac.uk/digisys/The_microbian_project

[3] Micro:bit Educational Foundation. (2023). *Hardware - Micro:bit*. Obtenido de https://tech.microbit.org/hardware/2-0-revision/

[4] Nordic Semiconductor. (2021). 6.18 RADIO - 2.4 GHz radio. En *nRF52833 Product Specification* (pág. 280). Obtenido de https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[5] Lancaster University. (s.f.). *Micro:bit Runtime*. Obtenido de https://lancaster-university.github.io/microbit-docs/#introduction

[6] Herrada, E. (2023). *Adafruit_Microbit*. Obtenido de GitHub: https://github.com/adafruit/Adafruit_Microbit

[7] Van Heesh, D. (28 de Junio de 2023). *Doxygen*. Obtenido de https://www.doxygen.nl/index.html

[8] Micro:bit Help & Support. (22 de Junio de 2023). *Micro:bit Error Codes*. Obtenido de https://support.microbit.org/support/solutions/articles/19000016969-micro-bit-error-codes

[9] Free Software Foundation, Inc. (2023). Obtenido de Using The GNU Compiler Collection (GCC): https://gcc.gnu.org/onlinedocs/gcc-13.1.0/gcc/#toc-GCC-Command-Options

[10] Nordic Semiconductor. (2021). 6.27.8.22 ADDRESS. En *nRF52833 Product Specification* (pág. 443). Obtenido de https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[11] STMicroelectronics. (30 de Agosto de 2022). Register Description - CTRL_REG1_A (20h). Tablas 33 y 34. En *LSM303AGR - Ultra-compact high-performance eCompass module: ultra-low power 3D accelerometer and 3D magnetometer* (pág. 47). Obtenido de https://www.st.com/resource/en/datasheet/lsm303agr.pdf

[12] STMicroelectronics. (2022). 4.2. Accelerometer - 4.2.1 Accelerometer power modes. Tabla 14. En *LSM303AGR - Ultra-compact high-performance eCompass module: ultra-low power 3D accelerometer and 3D magnetometer* (pág. 27). Obtenido de https://www.st.com/resource/en/datasheet/lsm303agr.pdf

[13] STMicroelectronics. (2022). Register Description - CTRL_REG1_A (20h). Tabla 35. En *LSM303AGR - Ultra-compact high-performance eCompass module: ultra-low power 3D accelerometer and 3D magnetometer* (pág. 47). Obtenido de https://www.st.com/resource/en/datasheet/lsm303agr.pdf

[14] Micro:bit Educational Foundation. (2023). *Schematics*. Obtenido de Micro:bit Developer Community: https://tech.microbit.org/hardware/schematic/#v2-pinmap

[15] Suits, B. H. (2023). *"Frequencies of Musical Notes, A4 = 440Hz"*. Obtenido de Physics of Music - Notes: https://pages.mtu.edu/~suits/notefreqs.html

[16] Nordic Semiconductor. (2021). 6.8.2.10 PIN_CNF[n] (n=0..31). En *nRF52833 Product Specification* (págs. 147-148). Obtenido de https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[17] Nordic Semiconductor. (2021). 6.8.2.2 OUTSET. En *nRF52833 Product Specification* (pág. 145). Obtenido de https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[18] Nordic Semiconductor. (2021). 6.8.2.3 OUTCLR. En *nRF52833 Product Specification* (pág. 145). Obtenido de https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[19] Nordic Semiconductor. (2021). 6.26 TEMP - Temperature Sensor. En *nRF52833 Product Specification* (pág. 425). Obtenido de https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf

[20] Gardner, M. (Octubre de 1970). "The fantastic combinations of John Conway's new solitaire game 'life'". (S. American, Ed.) *Mathematical Games, 223*(4), 120-123. Obtenido de https://web.stanford.edu/class/sts145/Library/life.pdf

A. Fichero de cabecera de la librería Ubit (*ubit.h*)

```
#include "../microbian/microbian.h"
#include "../microbian/hardware.h"
#include "../microbian/lib.h"

/* misc.c */
void main(int n);
void microbit_inicializa_hardware();
float termometro_lectura();

/* display.c */
#define DISPLAY_DIM 5
image imagen_actual_microbian;
typedef enum {INT_BAJA, INT_MEDIA, INT_ALTA} intensidad_t;
typedef enum {LENTO, MEDIO, RAPIDO} velocidad_texto_t;
typedef int imagen_t[DISPLAY_DIM][DISPLAY_DIM];
int display_cambia_intensidad(intensidad_t i);
int display_enciende_LED(int x, int y);
int display_apaga_LED(int x, int y);
int display_muestra_imagen(imagen_t img);
int display_muestra_secuencia(imagen_t seq[], int n_elem_seq, int delay_ms);
void display_limpia();
int display_muestra_sprite(char *sprite_bin);
void display_char2codi(char c, char **codi);
void display_muestra_texto(char *str, velocidad_texto_t v);

/* botones.c */
typedef enum {BOTON_A, BOTON_B, BOTON_LOGO, BOTON_0, BOTON_1,
BOTON_2} boton_t;
int boton_pulsado(boton_t b);
int boton_espera_pulsacion(boton_t b);

/* acelerometro.c */
void acelerometro_inicializa();
int acelerometro_lectura_x();
int acelerometro_lectura_y();
int acelerometro_lectura_z();
float acelerometro_inclinacion_eje_x();
float acelerometro_inclinacion_eje_y();
void brujula_inicializa();
int brujula_lectura_x();
int brujula_lectura_x2();
int brujula_lectura_y();
int brujula_lectura_z();

/* buzzer.c */
#define NEGRA       1000
#define CORCHEA     500
#define SEMICORCHEA 250
typedef enum {DO_4, RE_4, MI_4, FA_4, SOL_4, LA_4, SI_4, DO_5, RE_5, MI_5,
FA_5, SOL_5, LA_5, SI_5, DO_6} nota_t;
void buzzer_reproduce_nota(nota_t n, int t_ms);
```