Facultad
de
Ciencias

Portado del Sistema Operativo M2OS al microcontrolador "Micro:bit"

Porting the M2OS Operating System to the "Micro:bit" microcontroller

Trabajo de Fin de Grado
Grado en Ingeniería Informática

Autor: Mario Vicente García
Director: Mario Aldea Rivas
Co-Director: Héctor Pérez Tijero
Julio 2022

Resumen.

En la actualidad la empresa AdaCore, desarrolladora del compilador GNAT para el lenguaje Ada, incluye el micro:bit entre los microcontroladores soportados por su compilador. Además, AdaCore también proporciona soporte para micro:bit en la librería Ada Drivers Library. El micro:bit está basado en un microcontrolador ARM Cortex-M0 con 256KB de memoria flash y 16KB de RAM.

Sin embargo, el soporte que ofrece AdaCore para micro:bit tiene una limitación, el reducido número de tareas que es posible ejecutar de forma simultánea. Dado que el micro:bit solo tiene 16KB de RAM y que el tamaño de pila por defecto sugerido por GNAT es de 4KB, no es posible crear más de tres tareas.

Una opción ara intentar paliar esta limitación que nos aparece utilizando el soporte oficial de AdaCore es utilizar M2OS. M2OS es un sistema operativo desarrollado por el grupo de Ingeniería Software y Tiempo Real del departamento de Ingeniería Informática, y, lo que más nos interesa de este sistema operativo es que implementa una política de planificación que permite que las pilas de todas las tareas del sistema compartan el mismo área de memoria. Este ahorro de memoria permitirá superar la limitación en el número de tareas que adolece el soporte original proporcionado por el compilador GNAT.

En este proyecto se ha realizado el portado del sistema operativo M2OS al micro:bit. Para comprobar que el portado es correcto se ejecutaron los ejemplos proporcionados por AdaCore sobre M2OS y, además, para comprobar que podemos incrementar el número de tareas que podemos utilizar de forma simultanea, se desarrollará una librería para el robot Maqueen de DFRobot para utilizar algunos de sus dispositivos integrados como sensores sigue lineas, motores, luces etc... en un programa concurrente implementado mediante tareas.

Palabras clave.

Microbit, Portar, DFRobot Maqueen, M2OS, Ada, Robótica.

Abstract.

Currently the company AdaCore, developer of the GNAT compiler for the Ada language, includes the micro:bit among the microcontrollers supported by its compiler. In addition, AdaCore also provides micro:bit support in the Ada Drivers Library. The micro:bit is based on an ARM Cortex-M0 microcontroller with 256KB of flash memory and 16KB of RAM.

However, this support offered by AdaCore for micro:bit has a limitation, the small number of tasks that can be created with it. Since the micro:bit only has 16KB of RAM and the default heap size suggested by GNAT is 4KB, it is not possible to create more than three tasks.

An option to try to alleviate this limitation that appears using the official support of AdaCore is to use M2OS. M2OS is an operating system developed by the Software Engineering and Real Time group of the Computer Engineering department, and what interests us most about this operating system is that it implements a scheduling policy that allows the stacks of all the system tasks share the same memory area. This memory saving will allow to overcome the limitation in the number of tasks that suffers the original support provided by the GNAT compiler.

To test that the porting of M2OS is correct, we executed the examples provided by AdaCore on M2OS and, in addition, to verify that we can increase the number of tasks that we can use simultaneously, we developed a library for the DFRobot Maqueen robot to use some of its integrated devices such as line follower sensors, motors, lights etc. implemented concurrently through multiple tasks.

Keywords.

Microbit, Port, DFRobot Maqueen, M2OS, Ada, Robotics.

Índice

Índice de figuras

1. Introducción y objetivos

1.1. Motivación.

Cada vez es más común utilizar microcontroladores embebidos para la gestión de múltiples aspectos de nuestra vida cotidiana, por ejemplo, en la domótica de nuestro hogar. Por otro lado, los microcontroladores de bajo coste cada vez son más populares, se trata de pequeños ordenadores capaces de ejecutar las órdenes grabadas en su memoria. Actualmente se encuentran en auge ya que son baratos y fácilmente disponibles para los aficionados y que cuentan con grandes comunidades en línea para ciertos procesadores. Además, a medida que avanza la tecnología, se requiere que estos microcontroladores ejecuten cada vez más tareas de forma simultánea. Entre estos microcontroladores se encuentra el que usaremos a lo largo del proyecto, denominado micro:bit.

El Micro Bit (también conocido como BBC Micro Bit, y, a su vez estilizado como micro:bit) es un sistema embebido de hardware libre basado en ARM. Está diseñado por la BBC en colaboración con hasta 29 empresas lideres en el sector tecnológico (Microsoft, Barclays, Samsung, Cisco etc.) con el fin de utilizarse para la educación informática en el Reino Unido.

La BBC planeó regalar este microcontrolador a todos los niños de 7 años en Gran Bretaña a partir del año 2015. Además, antes del lanzamiento, se puso a disposición un simulador en línea para ayudar a los educadores a prepararse para que pudiesen formar sin problemas sus alumnos. [1]

Después de la implantación exitosa de micro:bit en todo el Reino Unido, la BBC mostró el futuro de micro:bit y su adopción en otras partes del mundo.

Sin embargo, micro:bit tiene un problema, y es que, debido a las especificaciones técnicas y al soporte actual, nos impide el desarrollo de aplicaciones concurrentes complejas. Dado que el micro:bit solo tiene 16KB de RAM y que el tamaño de pila por defecto sugerido por GNAT es de 4KB, no es posible crear más de tres tareas en un mismo programa concurrente. Este aspecto representa una gran limitación a la hora de utilizar este microcontrolador en un entorno real.

Por otro lado, el soporte para el desarrollo de aplicaciones Ada en sistemas empotrados es todavía reducido. En el caso del micro:bit, el compilador de AdaCore y la librería 'Ada Drivers Library' [2] son sus principales exponentes. En esta librería podemos encontrarnos gran cantidad de ejemplos para ejecutar en el microcontrolador así como el código que permite la programación de los dispositivos integrados en el microcontrolador (temporizador, entradas/salidas, etc.)

La principal motivación de este proyecto viene dada por esta popularidad que obtuvo micro:bit en su lanzamiento, y, sobretodo, que no exista este soporte para aplicaciones concurrentes complejas.

Para intentar paliar con este límite, se propone portar el sistema operativo M2OS. Este sistema operativo fue desarrollado por la universidad de Cantabria y, gracias a su gestión de la memoria, permite desarrollar aplicaciones más complejas que las que pueden concebirse con el software original.

En la actualidad, existen diversas alternativas a Micro:bit como pueden ser Raspberry Pi, Arduino Uno, Parallax Basic Stamp, Calliope mini o Phidgets entre otras. Algunas de las características que más destacan del micro:bit son:

- Posee gran variedad dispositivos ya integrados en la propia placa para crear programas interesantes, como un conjunto de leds, un acelerómetro, una brújula, botones, etc...
- Ofrece la posibilidad de poder diseñar programas de una forma eficiente, sencilla y divertida.
- Permite cargar los programas generados al microcontrolador de una forma muy sencilla, basta con copiar el programa a su tarjeta como si fuese una memoria USB, sin configuraciones complicadas ni retrasos añadidos por el compilador.

1.2. Objetivos.

Este trabajo tiene como objetivo el portado del sistema operativo M2OS para el microcontrolador micro:bit para intentar paliar con la limitación de no poder ejecutar aplicaciones concurrentes complejas con el software oficial de AdaCore. El portado se realizará utilizando parte del código de bajo nivel proporcionado en la librería oficial de AdaCore 'Ada Drivers Library'. La razón de utilizar esta librería es que la experiencia adquirida en este proyecto facilitaría el portado de M2OS al resto de microcontroladores soportados por dicha librería.[2]

Como objetivos adicionales al anterior tenemos los siguientes:

- Realizar el portado de un conjunto de librerías y/o drivers que permitan realizar entrada/salida mediante los pines del micro:bit y controlar los dispositivos integrados (leds, botones, acelerómetro, bus I2C, etc.).
- Programar un conjunto de librerías y/o drivers que permitan controlar sensores y actuadores habituales en robótica/domótica (motores, sensores de distancia, sensores climáticos,etc.) utilizando el robot DFRobot Maqueen.
- Desarrollar un conjunto de programas de prueba para el robot Maqueen y que aproveche todos los dispositivos que incorpora.
- Desarrollar un demostrador consistente en una aplicación robótica de mediana complejidad que nos permita visualizar de forma efectiva como nos saltamos el limite de 3 tareas que nos establece AdaCore.

2. Herramientas, tecnologías y métodos

2.1. Tecnologías y lenguajes.

2.1.1. Micro:bit

Micro:bit V1 (Figura 1) está formado por una placa Nordic nRF51822-QFAA-R rev 3 la cual incluye un procesador ARM Cortex-M0 de 32 bit. Esta primera versión, es una placa perfecta para comenzar en el mundo de la programación y posee algunas características interesantes como las detalladas a continuación:

- Bajo coste. La placa se encuentra en la franja de los 20€, precio muy asequible para la inicialización de los programadores en esta plataforma.
- Kits de desarrollo. Existen numerosos kits compatibles con la placa que nos permiten incrementar el número de periféricos con los que poder realizar gran cantidad de proyectos interesantes.
- Compatibilidad. Sus características la hacen compatible con una gran cantidad de dispositivos como por ejemplo, el robot Maqueen de DFRobot que utilizaremos en este trabajo.
- Facilidad de uso. Como ya se ha comentado previamente, al ser una placa diseñada para la enseñanza, posee gran cantidad de librerías o programas con los que experimentar incluso de forma online. Por lo tanto, incluso para los más novatos, implementar sus pruebas y prototipos no requerirá mucho esfuerzo.

La placa micro:bit dispone de 19 pines de los cuales, 3 pueden ser utilizados como salidas PWM (para reproducir sonidos por un zumbador o controlar servomotores), 6 pines pueden ser asignados como entradas analógicas (para detectar valores de diferentes sensores). Además, todos los pines pueden ser asignados para entrada salida digital, 2 de ellos son utilizados para la detección de los botones, 2 son asignados para la interfaz I2C (Para comunicarnos con diferentes dispositivos), posee además una brújula, una entrada USB que sirve tanto para alimentación como para transferir programas a la placa y un botón reset.

La placa posee también una matriz de leds de 5x5 de color rojo con 10 posibles valores de intensidad.

La memoria Flash que dispone la placa es de 16Kb junto con una memoria ROM de 256Kb, además, micro:bit funciona a una frecuencia de reloj de 16 MHz.

Esta placa funciona con una alimentación externa de entre 1.8 a 3.6 voltios. La alimentación podrá ser obtenida a través del conector USB o con una fuente de alimentación externa formada por 2 pilas AAA.

Por último, el micro:bit nos permite una comunicación inalámbrica vía bluetooth a una frecuencia de 2.4 GHz lo que nos otorga posibilidades como comunicarse con gran variedad de dispositivos como por ejemplo smartphones y tablets.[3]

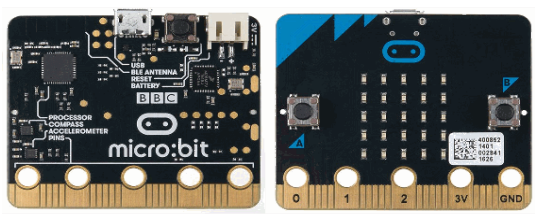

Figura 1: Microcontrolador micro:bit

2.1.2. Ada

Ada es un lenguaje de programación estático y fuertemente tipado. Fue diseñado por un equipo dirigido por Jean Ichbiah de CII Honeywell Bull tras un encargo del Departamento de Defensa de los Estados Unidos cuyo principal objetivo era prescindir de los cientos de lenguajes que utilizaban y conseguir uno especializado que fuese capaz de realizar o incluso mejorar las tareas de todos ellos.

Ada es un lenguaje concurrente, multipropósito y orientado a objetos con unos valores fundamentales que se apoyan en la eficiencia, seguridad, mantenibilidad y facilidad de uso. Hoy en día, Ada es utilizado tanto para la gran mayoría de hardware militar como en el campo de la aeronáutica, industria aeroespacial y todo aquel sistema crítico en el que cualquier fallo en el software puede traer graves consecuencias para personas y/o infraestructuras.

El compilador utilizado para Ada es GNAT. Este compilador fue desarrollado por la Universidad de Nueva York y también patrocinado por el Departamento de Defensa de EE.UU. Está basado en GCC y es software libre. Actualmente está mantenido por la empresa AdaCore.[4]

Figura 2: Logo lenguaje Ada

2.1.3. Soporte de AdaCore para micro:bit

Como se ha mencionado en un apartado previo, AdaCore ofrece soporte oficial para el microcontrolador micro:bit. Este soporte que ofrece lo podemos encontrar en dos productos:

- La versión "ARM ELF" del compilador GNAT.[5] Es una versión cruzada del compilador que ejecuta sobre Linux y permite generar código para placas con procesadores ARM. Esta versión del compilador incluye las librerías de tiempo de ejecución (Run-time System library, RTS) para las plataformas ARM soportadas (entre ellas el microcontrolador micro:bit). Para cada uno de estos microcontroladores compatibles se incluyen tres versiones del RTS:
  - ZFP (Zero Footprint). No soporta tareas.
  - Ravenscar SFP (Ravenscar Small Footprint). Soporta tareas pero no soporta propagación de excepciones (una versión simplificada de este RTS es usada por el sistema operativo M2OS).
  - Ravenscar Full. Soporta tareas y propagación de excepciones.
- Librería "Ada Drivers Library". Es una librería completa que nos incluye código relacionado con la inicialización y la gestión del hardware para multitud de microcontroladores, además, incluye una gran variedad de manejadores de dispositivos.

Sin embargo, el soporte oficial de AdaCore no nos permite ejecutar gran cantidad de tareas en un mismo programa concurrente, ya que, independientemente de la tarea que estemos ejecutando, el espacio que se reserva en memoria para esta es el mismo. Si a este problema le sumamos la escasa memoria que incorporan la mayoría de microcontroladores, nos aparece este problema de poder ejecutar únicamente tres tareas en una aplicación paralela en el caso de micro:bit.

2.1.4. Robot Maqueen

DFRobot[6] se fundó a partir de una comunidad de fabricantes locales en 2008, DFRobot fue de las primeras empresas en adoptar hardware de código abierto. Actualmente siguen creando productos de hardware/software innovadores y fáciles de usar que se convierten en los componentes básicos de todo tipo de proyectos electrónicos y que fomentan una comunidad sólida de estudiantes.

Maqueen(Figura 3) es un robot educativo y de programación sencilla diseñado para la educación, Maqueen recibe la jugabilidad y la operación simple de micro:bit. Es un juguete perfecto para que los niños comiencen a aprender robótica ya es compatible con Makecode, Scratch y python, por lo que iniciarse a la programación con este robot será una tarea sencilla. Esta robot ofrece un tamaño pequeño y un movimiento flexible gracias a sus ruedas, lo que facilita su instalación y su uso. Este robot incluye sensores que admiten la interacción de sonido, luz o incluso señales inalámbricas.

Figura 3: DFRobot Maqueen

Maqueen esta diseñado para conectar de forma directa el microcontrolador micro:bit, además, nos permite utilizar los pines uno, dos y tres de forma libre, ya sea para conectarle un sensor o bien para conectarle cualquier otro dispositivo, ya que, el conector de enganche que posee micro:bit con dispositivos nos bloquea/oculta el resto de pines, que, serán utilizados (algunos de ellos) por el propio dispositivo al que le conectemos.

Para la alimentación del Maqueen, se utiliza una alimentación externa conectada a al puerto específico del Maqueen, el cual también aporta energía al microcontrolador micro:bit conectado. Este conector para la alimentación que posee el robot Maqueen tiene un rango operativo de entre 3,5V y 5V y se utiliza una batería de tres pilas AAA. Un detalle importante es que nos debemos asegurar de no exceder los 5 V de alimentación, puesto que un mayor voltaje puede dañar el robot.

Por último, mirando desde un punto mas arquitectural, el robot Maqueen dispone de dos sensores sigue lineas para que nuestro robot siga una ruta dibujada, un zumbador para emitir sonidos, luces led en el frontal, luces RGB... entre otros elementos(Figura 4). Maqueen posee también dos motores para el movimiento de las ruedas a una velocidad máxima de 133rpm y, para la comunicación con ellos, se utilizará el protocolo I2C.

Este robot será importante en la recta final del proyecto ya que, al tener más elementos programables (Luces, motores, sensores...) permitirá crear una aplicación concurrente compleja a modo de demostrador final del trabajo.

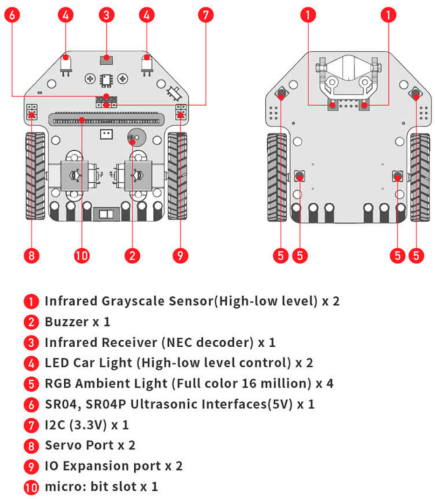

Figura 4: Especificaciones técnicas Maqueen

2.1.5. Librerías de soporte al proyecto

A lo largo de este proyecto se han utilizado las siguientes librerías:

- Ada Drivers Library[2]: Este repositorio contiene drivers y proyectos de muestra para programar microcontroladores con los lenguajes Ada y SPARK. La biblioteca también proporciona algunos servicios de middleware y controladores para dispositivos externos como sensores.

  El objetivo de este repositorio es crear un lugar para que tanto AdaCore como la comunidad en general aporten soporte para procesadores, plataformas y otros dispositivos adicionales. En el proyecto utilizaremos esta librería para tomar como referencia como configurar el timer de micro:bit, utilizar sus ejemplos corriendo sobre M2OS y para utilizar algunas de sus funciones como transmitir por el bus I2C.

- Librería Maqueen para Arduino[7]: Se trata de una librería simple para el robot DF Robot Maqueen, utilizando Arduino. Esta librería la tomaremos como referencia para realizar nuestra propia librería para Maqueen utilizando el microcontrolador micro:bit.

- Librería Baremetal[8]: Este repositorio contiene códigos compatibles con el chip nRF51822, el cual lleva la versión 1 de nuestro micro:bit. Esta librería será utilizada específicamente para realizar un análisis del bus I2C para comprobar cual es la dirección de los motores del robot Maqueen.

Estas librerías suponen un aspecto importante a lo largo del proyecto, la más importante es la librería 'Ada Drivers Library' ya que es la librería que nos va a incorporar toda la comunicación que tengamos con el microcontrolador en cada uno de los ejemplos que se implementen.

2.1.6. Protocolo I2C

I2C es un protocolo de comunicación en serie, define la trama de datos y las conexiones físicas para transferir bits entre 2 dispositivos digitales. La comunicación se realiza a partir de dos cables, SDA y SCL. Además el protocolo permite conectar hasta 127 dispositivos esclavos con esas dos líneas, con velocidades variables que rondan los 100, 400 y 1000 kbits/s.

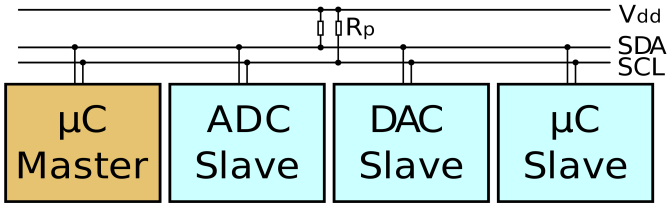

Figura 5: Esquema bus I2C

El protocolo I2C es uno de los más utilizados para comunicarse con sensores digitales ya que su arquitectura permite tener una confirmación de los datos recibidos, dentro de la misma trama, entre otras ventajas. Otra de sus principales ventajas es que podemos tener múltiples dispositivos conectados en el mismo bus IC.

Los mensajes enviados mediante I2C, incluyen, además del byte de información, la dirección del registro como la dirección del registro que se desea leer o escribir. Para la información que se envía siempre existe una confirmación de recepción por parte del dispositivo. Por esta razón es bueno diferenciar a los distintos elementos involucrados en este tipo de comunicación.[9]

El bus I2C será utilizado en el proyecto para realizar la comunicación con los motores del robot Maqueen.

2.2. Herramientas.

2.2.1. GNAT Programming Studio

GNAT Programming Studio o GNAT Studio es un potente IDE que da soporte tanto para la programación, como para debugging y el análisis de código. Además, nos ofrece una herramienta para automatizar la carga de aplicaciones en microcontroladores como Arduino uno, micro:bit..

Una de las principales características de GNAT Studio es que se trata de un IDE multilenguaje que da soporte a Ada, SPARK, C, C++ y Python. Se diseñó con la idea de dar versatilidad al usuario para permitirle poder adaptarse a todo tipo de proyecto y de cualquier tipo de complejidad.

Otra característica interesante de este IDE es que nos ofrece la posibilidad de extensión a través de scripting. Permite personalizar el entorno de trabajo en función de los que el usuario necesite de una forma sencilla. GNAT Studio también ofrece herramientas que facilitan el desarrollo cruzado de aplicaciones, esta técnica nos optimiza el desarrollo de una determinada aplicación en un computador y ejecutarlo en otro habitualmente menos potente.[10]

El IDE de GNAT utiliza ficheros con extension '.gpr'. Gracias a estos ficheros .gpr, GNAT nos permite administrar compilaciones complejas que involucran una gran cantidad de código fuente, diferentes directorios y opciones para diferentes configuraciones del sistema. En particular, estos ficheros '.gpr' nos permiten especificar propiedades como por ejemplo:

- Establecer el directorio o conjunto de directorios que contienen los archivos/códigos de origen.
- Indicar el directorio en el que se colocarán los programas ejecutables.
- Posibilidad de generar bibliotecas automáticamente como parte del proceso de compilación.

Estos ficheros de proyecto del IDE de GNAT están escritos en una sintaxis similar a Ada, utilizando nociones familiares como paquetes, cláusulas de contexto, declaraciones, valores predeterminados, asignaciones y herencias.

Además, estos ficheros '.gpr' ofrecen características como que puedan depender de otros ficheros '.gpr' de forma modular, lo que simplifica la integración de sistemas complejos y la reutilización de proyectos.

2.2.2. Cutecom

CuteCom es una terminal gráfica serie, como minicom o Hyperterminal en Windows. Actualmente es compatible con Linux, FreeBSD y MacOS X.

Cutecom está dirigido principalmente a desarrolladores de hardware u otras personas que necesitan un terminal para comunicarse con dispositivos. Es de software libre y se distribuye bajo Licencia Pública General. (GNU Versión 2)

Este proyecto utiliza dicho software para observar el resultado de ejecutar el código que realiza un escaneo del bus I2C de nuestro microcontrolador perteneciente a la librería baremetal y, de esta forma, poder ver todas las direcciones disponibles de nuestro microcontrolador micro:bit en el bus I2C.[11]

2.3. Sistema operativo M2OS

2.3.1. Introducción al sistema operativo.

M2OS es un pequeño sistema operativo de tiempo real que permite ejecutar aplicaciones multitarea con numerosas tareas utilizando el lenguaje Ada sobre microcontroladores con prestaciones de memoria muy limitadas (como es el caso de micro:bit). M2OS ha sido desarrollado por el departamento de Ingeniería Informática y Electrónica de la Universidad de Cantabria.[12][13]

La principal característica de este sistema operativo es que implementa una planificación no expulsora que permite un alto aprovechamiento de la memoria. Específicamente, M2OS nos permite que la zona de memoria asignada a la pila pueda ser compartida por todas las tareas activas del sistema, con lo cual, el sistema operativo solo necesita reservar un área igual al mayor tamaño de pila necesario de entre todas las tareas. De esta forma, se puede conseguir sobrepasar el número de tareas máximas que se pueden ejecutar de forma paralela en cualquier microcontrolador.

Esta característica de M2OS será en la que más se profundice a lo largo del proyecto ya que es la que nos va a permitir programar una aplicación concurrente con cierto grado de complejidad utilizando el microcontrolador micro:bit. Para entender mejor esta característica de como M2OS aprovecha la memoria, observemos en la figura 6.

Figura 6: Reserva de memoria sin M2OS

En la ilustración podemos ver una barra la cual representa toda la memoria disponible en el microcontrolador micro:bit (16Kb), en la figura 7, podemos ver cuatro tareas independientes pero del mismo programa. Si ejecutamos este programa sin M2OS, por cada tarea existente se reservará cierto espacio en memoria, mientras que, si ejecutamos el mismos programa en M2OS, se gestionará la memoria de tal forma que únicamente será necesario reservar memoria para la tarea que más memoria necesite reservar, en el caso del ejemplo, la tarea 1.

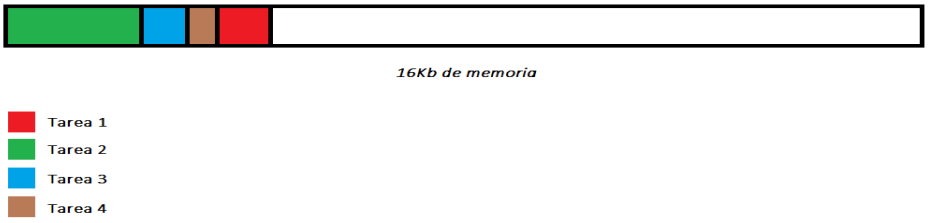

Figura 7: Reserva de memoria con M2OS

En la actualidad M2OS se encuentra portado a tres arquitecturas:

- Arduino Uno
- STM32F4
- Many-core Epiphany

La arquitectura stm32f4 toma una gran importancia en nuestro portado al microcontrolador micro:bit ya que, ambos utilizan un procesador Arm Cortex. En este proyecto incluiremos una nueva arquitectura denominada "Microbit"

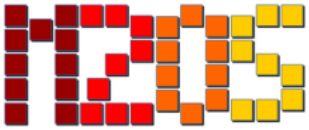

Figura 8: Logo M2OS

2.3.2. Estructura del sistema operativo

A continuación se describen las carpetas que componen la distribución de M2OS y se comenta brevemente su función dentro del sistema operativo:

- wiki/: Aquí tenemos toda la documentación de M2OS, se genera automáticamente a la hora de crear la distribución a partir de la wiki del GitLab de M2OS.
- tests/: Aquí podemos encontrar tests funcionales, de prestaciones y de tamaño. Contiene dos carpetas, api_m2os/ (incluye los tests que utilizan la API de M2OS para crear tareas) y ada_tasks/ (incluye los tests que utilizan tareas Ada).
- scripts/: scripts Python que definen los botones de ejecución en la placa y en el emulador para la arquitectura 'arduino_uno'.
- m2os_tool/: Se trata de una herramienta de transformación automática de código que permite transformar código multitarea escrito en lenguaje Ada estándar (utilizando la palabra reservada Task) a código que utiliza la API de M2OS para crear las tareas.
- kernel/: Aquí encontramos el código de la parte de M2OS no dependiente del hardware.
- gnat_rts/: Run-time System para las distintas arquitecturas. En nuestro caso utilizaremos el mismo RTS que en la arquitectura stm32f4.
- examples/: En esta carpeta tenemos ejemplos de aplicaciones generales y específicas para cada arquitectura. En nuestro caso crearemos más adelante ejemplos para la arquitectura Microbit.
- arch/: Aquí residen carpetas con el código específico de cada arquitectura. Cada carpeta correspondiente a cada arquitectura contiene, al menos, las carpetas hal/ y drivers/. También incluye los ficheros de proyecto específicos de la arquitectura.
- adax/: Directorio donde residen las extensiones a los paquetes estándar Ada definidas para M2OS.

En la raíz del sistema operativo tenemos una serie de ficheros importantes para la instalación del mismo, los ficheros se detallan a continuación:

- config_params.mk: indica la arquitectura que se desea instalar y las opciones de compilación (optimización y depuración) del código del usuario y del sistema operativo. Es editado por el usuario.
- rules.mk: comandos de compilación de ficheros en los diferentes lenguajes y reglas para el borrado de ficheros generados durante la instalación.
- config.mk: parámetros generales. Muchos de los parámetros de este fichero no se utilizan en el proceso actual de instalación por lo que necesitaría un limpiado y reorganización (pero dichas tareas quedan fuera de los objetivos de este proyecto).
- Makefile: realiza la instalación de M2OS para la arquitectura seleccionada. Incluye los ficheros config_params.mk, rules.mk y config.mk.

3. Portado del sistema operativo M2OS a la placa.

Como se comentó en la sección de objetivos, en este proyecto se ha buscado portar el sistema operativo M2OS al microcontrolador micro:bit, lo que permitirá ejecutar aplicaciones Ada con gran número de tareas en un mismo programa concurrente.

El portado a una nueva plataforma implica cuatro tareas: implementación de la interfaz abstracta con el hardware, adaptación de la librería de tiempo de ejecución del lenguaje Ada (Run-Time System, RTS), desarrollo de drivers de dispositivos e integración en el sistema de instalación y compilación de M2OS. Una de las dificultades del portado es que las cuatro tareas deben estar casi completas para poder realizar las primeras pruebas. Puede ser posible realizar pruebas parciales disponiendo únicamente de un driver simple (consola o leds). También pueden realizarse pruebas preliminares utilizando un RTS sin soporte para tareas (denominado Zero Foot Print, ZFP). Además, se pueden comenzar las pruebas sin tener la interfaz abstracta con el hardware completamente implementada (por ejemplo sin tener aún soporte para interrupciones y timer).

Antes de explicar el proceso de portado del sistema operativo, es necesario tener una visión global de la arquitectura sobre la que estamos trabajando:

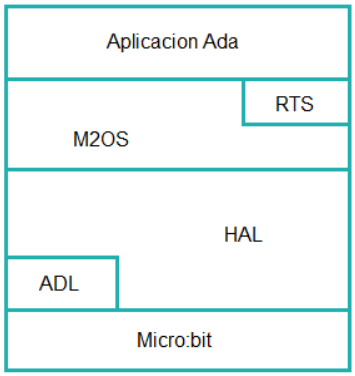

Figura 9: Diagrama de Capas de la arquitectura

En el diagrama superior vemos como se comunican las distintas partes que componen la arquitectura y, por ende, este proyecto. La aplicación de alto nivel escrita en lenguaje Ada, se comunicará directamente con el sistema operativo M2OS, al compilar, hará uso del RTS (Run-Time System). El RTS aporta a las aplicaciones Ada el soporte para la multitarea. Además incluye los paquetes de utilidad definidos por el lenguaje (gestión del tiempo, soporte matemático, estructuras de datos, etc.). El compilador enlaza el RTS con el código del usuario para construir el fichero binario que es cargado y ejecutado en el microcontrolador.

M2OS tiene una capa de interfaz abstracta con el hardware (Hardware Abstract Layer, HAL) que implementa toda la interacción con el hardware requerida para el funcionamiento del sistema operativo. El HAL de M2OS se implementa en el fichero M2.HAL (ficheros m2-hal.ads y m2-hal.adb). Para el portado a micro:bit será necesario crear los ficheros microbit/hal/m2-hal.ads y microbit/hal/m2-hal.adb.

Por otro lado, tanto los programas Ada de alto nivel como el propio HAL se comunicarán con la librería 'Ada Drivers Library', la cual se integrará en el propio sistema operativo M2OS. De esta forma, se podrán utilizar paquetes ya programados por AdaCore para la comunicación con la placa y utilizar drivers para utilizar los dispositivos integrados en el microcontrolador, utilizar el protocolo I2C...

Una vez que el compilador enlaza el RTS con el código del usuario para construir el fichero binario se procederá a cargar dicho binario en el microcontrolador para su ejecución.

3.1. Preparación del entorno

A lo largo de este proyecto utilizamos el GNAT Studio Programming en su versión correspondiente al año 2021, aunque es posible utilizar el programa de GNAT en cualquiera de sus últimas versiones.

Además, para compilar nuestros programas será necesario la versión 'ARM ELF' del compilador GNAT. La versión que se utilizará será la del año 2018 por ser la soportada por M2OS, esta versión incluye las librerías de tiempo de ejecución para las plataformas ARM soportadas (entre otras el microcontrolador micro:bit).

En el caso de no tener previamente el IDE de GNAT o la versión del compilador 'ARM ELF' instalados se debe proceder a su instalación. El anexo A desarrolla el proceso que se debe llevar a cabo para este objetivo, mientras que, en el anexo B se puede ver el proceso de instalación para la librería 'Ada Drivers Library'.

Para comprobar el correcto funcionamiento del compilador y del IDE se procedió a la ejecución de un programa de prueba. Para ello, se compiló el ejemplo perteneciente a librería 'Ada Drivers Library' denominado 'Text Scrolling' el cual mostrará un texto por los leds de nuestro micro:bit.

Dentro del directorio de la libreria 'Ada Drivers Library' se ejecutó el comando para abrir el IDE de GNAT (gnatstudio -P) cargando a su vez el proyecto correspondiente al ejemplo concreto 'Text Scrolling'. El comando ejecutado fue el siguiente:

```
$ gnatstudio -P examples/MicroBit/text_scrolling/text_scrolling.gpr &
```

Una vez abierto el proyecto, se abrirá la interfaz de GNAT Studio Programming como se puede ver en la figura 10.

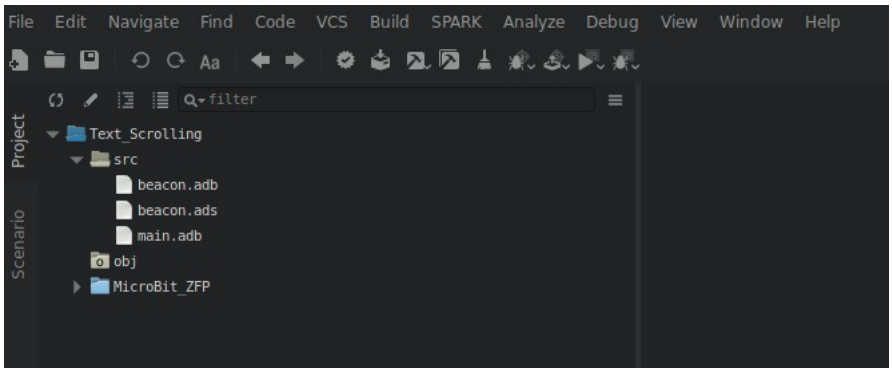

Figura 10: Interfaz de GNAT Programming Studio

Para cargar y ejecutar el programa en nuestro microcontrolador, simplemente habrá que tener la placa micro:bit conectada al computador y, posteriormente, generar el binario para cargar en la placa. El botón de cargar en la placa tendrá el mismo funcionamiento que el resto de botones juntos, es decir, comprobará sintaxis, compilará todo el código del proyecto y, finalmente, lo cargará en la placa.

Un sistema empotrado como el micro:bit no dispone de una pantalla para ver el resultado de la ejecución de nuestro programa, por lo que la salida de la consola se debe de obtener de otra forma. En este caso obtendremos la salida a través del puerto serie.

Un puerto serie es una interfaz de comunicación a través de la cual la información se transfiere de forma bidireccional y de forma secuencial pero con la característica de que únicamente se transmite un bit a la vez. Esto contrasta con un puerto paralelo, que envía múltiples bits de forma simultanea.

Para leer el puerto serie del dispositivo, linux nos ofrece un comando denominado stty, este es un comando que se usa para imprimir por una terminal lo que se transmita por el puerto serie. En resumen, este comando muestra o cambia las características del terminal.

El comando utilizado fue el siguiente:

```
$ stty -F /dev/ttyACM0 cs8 115200 ignbrk -brkint -icrnl
-imaxbel -opost -onlcr -isig -icanon -iexten -echo -echoe
-echok -echoctl -echoke noflsh -ixon -crtscts && cat /dev/ttyACM0
```

Con este comando se podrá leer el dispositivo correspondiente al microcontrolador micro:bit y ver su salida. El flag -F es utilizado para indicar el dispositivo del que se quiere leer el puerto serie y después una serie de opciones como colocar el puerto serie a 115200 baudios y otros flags de configuración que pueden consultarse en el manual del comando stty. Además se utiliza el comando 'cat' para leer el 'fichero' que se corresponde con el dispositivo micro:bit, para poder visualizar su puerto serie una vez configurado.

La librería 'Ada Drivers Library' proporciona numerosos ejemplos con los que poder probar diversas funcionalidades que incorpora el micro:bit como puede ser el giroscopio, los botones etc...

3.2. Introducción al portado

Antes de comenzar con el portado, es necesario explicar que la placa stm32f4, donde M2OS ya se encuentra portado, utiliza un procesador ARM Cortex-m4, mientras que, nuestro micro:bit, utiliza Cortex-m0. Ambos procesadores son procesadores ARM, sin embargo, el Cortex-m4 es un procesador más complejo que el Cortex-m0. Ambas arquitecturas utilizan una versión para microcontroladores denominada ARMv-M, las cuales son versiones especificas de ARM para microcontroladores, el Cortex-m4 utiliza la version ARMv7-M mientras que el procesador Cortex-M0 que integra el micro:bit utiliza la versión ARMv6-M.

Cabe destacar que la versión ARMv6-M está dirigido al extremo más bajo del espacio de microcontroladores de 32 bits, lo que permite diseños con un número muy bajo de puertas y que cuenta con una micro arquitectura muy simple y altamente eficiente. Para abaratar el precio y conseguir esta simplicidad, se eliminaron varias características de ARMv7-M. La figura 11 muestra los principales cambios.

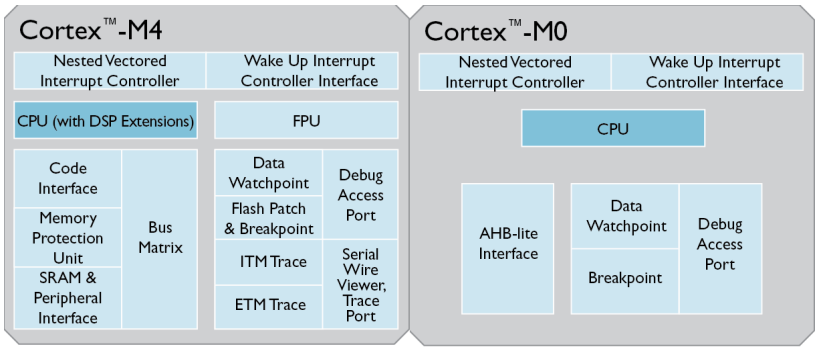

Figura 11: ARMv7-M vs ARMv6-M

Entre todas las características que diferencian a la arquitectura ARMv6-M con la arquitectura ARMv7-M cabe destacar las siguientes:

- El conjunto de instrucciones ARMv6-M es el más pequeño admitido por todos los procesadores ARM, con solo 57 instrucciones distintas.
- El temporizador SysTick es una opción de implementación.
- El número máximo de interrupciones externas está limitado a 32 y solo hay cuatro niveles de prioridad disponibles.
- Los accesos a la memoria siempre deben estar alineados de forma natural.

Debido a que ambos procesadores cuentan con arquitecturas muy similares, se utilizarán los ficheros de la arquitectura del stm32f4 como punto de partida. Los ficheros que se copiarán serán los que se sitúan en la carpeta arch/.

El primer objetivo del portado será hacer que la instalación de M2OS se ejecute correctamente con la arquitectura de microbit.

Para ello, lo primero que se hizo fue crear el directorio 'microbit' dentro del directorio arch/, la cual, como se explicó previamente, contiene el código específico de cada arquitectura con la que es compatible M2OS.

Dentro de esta nueva carpeta se crearon dos directorios nuevos, el directorio drivers/ y el directorio hal/. En el directorio drivers/ no fue necesario agregar ningún fichero, mientras que, en la carpeta hal, irán colocados los códigos específicos de la arquitectura (hal.adb y hal.ads) los cuales serán copiados de la arquitectura stm32f4. Además, dentro de la carpeta 'microbit' tendremos los ficheros de proyecto de GNAT específicos de la arquitectura: 'shared_switches_microbit.gpr' y 'm2os_microbit.gpr'.

En una primera instancia, no se editarán los ficheros correspondientes al hal, sin embargo, si que se editarán los ficheros del proyecto correspondientes a esta nueva arquitectura:

- m2os_microbit.gpr: Este fichero de proyecto del IDE de GNAT, establece toda la configuración base que tendrán todos los programas de M2OS que se ejecuten sobre micro:bit así como indicar directorios de compilación, librerías a utilizar, que compilador se va a utilizar etc...

  Como se parte del fichero base de la arquitectura stm32f4, en este fichero únicamente se editarán todas las referencias que se hagan a esta arquitectura y se reemplazarán por la nueva arquitectura 'microbit'. Además, se eliminó de los directorios fuente donde se buscan códigos fuentes, la carpeta /hal/stm32f4 ya que en el directorio del micro:bit no existe.

  Por otro lado, se eliminaron los paquetes que no tenemos disponibles en micro:bit, estas son:

  `"leds"`, `"Registers"`, `"stm32f4"`, `"stm32f4.gpio"`, `"stm32f4.reset_clock_control"` y `"stm32f4.sysconfig_control"`

- shared_switches_microbit.gpr: Este fichero, al igual que el fichero visto previamente, establece una serie de configuraciones que heredaran todos los proyectos de ejemplos de micro:bit. Algunas de las configuraciones que se indican en este fichero son:

  - Utilizar algunos flags específicos de compilación como '-ffunction-sections' y '-fdata-sections'.
  - Utilizar como herramienta de conexion 'pyocd', el protocolo a utilizar así como el puerto.
  - El emulador que se puede utilizar para cargar nuestros programas de forma virtual.

  Las modificaciones realizadas sobre este fichero de proyecto del IDE de GNAT será reemplazar todas las coincidencias de la palabra stm32f4 por microbit.

3.3. Modificación de ficheros para la instalación de la arquitectura

Ahora que se tiene la base de ficheros para realizar el portado, el siguiente paso será hacer que la instalación para la arquitectura de microbit se realice de forma exitosa.

Para elegir la arquitectura para la que se quiere instalar M2OS es necesario modificar el fichero 'config_params.mk' del directorio principal de M2OS. En este fichero deberemos de comentar todas las arquitecturas salvo la que estemos interesados en instalar.

```
#
# Target (only one line can be uncommented)
#
#CONFIG_BOARD_RPI (not supported)
#CONFIG_BOARD_ARDUINO=y
#CONFIG_BOARD_STM32F4=y
CONFIG_BOARD_MICROBIT=y
#CONFIG_BOARD_GNATBB (not supported)
#CONFIG_BOARD_EPIPHANY=y
#CONFIG_BOARD_AVR_IOT=y # (under development)
```

En este fragmento de código del fichero, como la única opción sin comentar es la del microbit, en el momento de la instalación del sistema operativo, se instalará esta nueva arquitectura. En este fichero también se pueden comentar o descomentar diferentes opciones de instalación como habilitar flags de Debug para M2OS, el RTS o las aplicaciones.

Una vez modificado este fichero, si se intenta realizar un primer intento de instalación de M2OS, comenzará la instalación pero se obtendrán errores, ya que, serán necesarias más modificaciones en distintos

ficheros para que este proceso de instalación se ejecute de forma correcta.

A raíz de los mensajes de error que retorna la instalación, se fueron solucionando los problemas existentes en la instalación hasta que el proceso finalizó de forma exitosa. Las modificaciones realizadas y sobre que ficheros se realizaron se detalla a continuación:

- En el fichero arch/microbit/hal/m2-hal.adb se eliminó la llamada a la función 'M2_HAL_Init_Timer'. Esta función realizaba la inicialización del timer en la arquitectura stmf32f4, sin embargo, en la arquitectura de microbit, esta inicialización se realiza en la función 'Initialize_board'.

- En el fichero 'kernel/m2-end_execution.adb' se añadirá código de soporte para poder apagar o reiniciar la placa, para ello, se añadirá la arquitectura 'microbit' en la siguiente condición:

```
...
#if M2_TARGET = "stm32f4" or M2_TARGET = "epiphany" or M2_TARGET = "microbit" then
pragma Import (Ada, Os_Exit, "_exit_gnat");
--  Shutdown or restart the board (Defined in System.Machine_Reset)
```

Una vez realizados estos dos pequeños cambios, si se intenta realizar de nuevo la instalación de M2OS para la arquitectura de micro:bit se obtendrá la siguiente salida:

```
gprinstall -p -f -P /home/mario/opt/GNAT/2018-arm-elf/arm-eabi/BSPs/m2os_microbit.gpr
Install project Install
Install project Libgnarl
Install project Libgnat
-- Building M2OS ...
(flags:  -Os -gnatp)
gprbuild -P /home/mario/Documentos/TFG/M2OS-microbit//arch/microbit/m2os_microbit.gpr
Compile
   [Ada]          m2-kernel-scheduler.adb
   [Ada]          m2-end_execution.adb
   [Ada]          m2-debug.adb
   [Ada]          m2.ads
   [Ada]          m2-kernel-initialization.adb
   [Ada]          m2-drivers.ads
   [Ada]          m2-kernel-api.adb
   [Ada]          m2-kernel.adb
   [Ada]          m2-kernel-tcbs.adb
   [Ada]          m2-kernel-timing_events.adb
   [Ada]          m2-kernel-ready_queue.adb
   [Ada]          m2-hal.adb
m2-hal.adb:137:10: warning: Stack management functions not implemented
   [Ada]          m2-direct_io.adb
   [Ada]          m2-kernel-dispatcher.adb
   [Ada]          m2-queue.adb
   [Ada]          m2-queue_implementation.adb
Build Libraries
   [gprlib]       m2os.lexch
   [bind SAL]     m2os
   [Ada]          b__m2os.adb
   [archive]      libm2os.a
   [index]        libm2os.a
DONE
M2OS is set to use the microbit architecture
```

Figura 12: Instalación de la arquitectura micro:bit exitosa

Lo que significa que la instalación para esta nueva arquitectura se realiza de forma correcta y sin ningún error.

Ahora que la instalación se ejecuta correctamente, se procedió a compilar y a cargar un primer ejemplo en la placa. Para conseguir este objetivo lo primero que se hizo fue seguir modificando todos los ficheros

de M2OS donde tengamos una ocurrencia de 'M2_TARGET = "stm32f4 para añadir el caso de micro:bit. Este cambio se realizó sobre los siguientes ficheros:

- adax/adax_dispatching_stack_sharing-periodic_task.adb (Línea 61)
- kernel/m2-kernel-scheduler.adb (Línea 94)
- tests/reports/measurements.adb (Línea 49)

3.4. Integración de la librería ADL al proyecto de micro:bit

Ahora que ya se instala M2OS correctamente para la arquitectura microbit, el siguiente paso será la integración de la librería 'Ada Drivers Library' en M2OS para utilizar muchos de los ejemplos y utilidades programadas que AdaCore ofrece con el fin de utilizar los dispositivos integrados en nuestro microcontrolador.

Para ello se debe descargar previamente la librería 'Ada Drivers Library' como se indica en el anexo B. Una vez descargada, Se debe de copiar la carpeta descargada en el directorio de M2OS 'arch/microbit'.

Después de situar la librería de AdaCore en arch/microbit, se creó el directorio examples/api_m2os/microbit/ para ir colocando en su interior los ejemplos que utilicen la librería 'Ada Drivers Library'. Además se copió uno de los ejemplos de la librería, específicamente 'leds_text_scrolling.adb' que ya usamos previamente.

Una vez creada la carpeta con el ejemplo, fue necesario crear un fichero de proyecto de tipo '.gpr' como los que se han visto previamente para que usase como directorio de recursos la propia librería de 'Ada Drivers Library', de esta forma, en todos los ejemplos que integren este proyecto, podremos utilizar códigos y drivers que nos ofrece dicha librería. Este proyecto se denominó examples/api_m2os/microbit/examples_microbit.gpr y, para incorporar las utilidades de la librería de AdaCore, se incorporó en la sección 'source_dirs' el siguiente fragmento de código:

```
...
   # Project source directories
   ADL_Root := Shared_Switches_Microbit'Project_Dir & "/Ada_Drivers_Library";
   for Source_Dirs use
     (".",
      "src",
      Global_Switches'Project_Dir & "adax",
      Global_Switches'Project_Dir & "arch/microbit/drivers/libcore",
      ADL_Root & "/hal/src/", -- From HAL config
      ADL_Root & "/boards/MicroBit/src/", -- From board definition
      ADL_Root & "/arch/ARM/cortex_m/src", -- From arch definition
      ADL_Root & "/arch/ARM/cortex_m/src/cm0", -- From arch definition
      ADL_Root & "/arch/ARM/cortex_m/src/nocache", -- From arch definition
      ADL_Root & "/arch/ARM/cortex_m/src/nvic_cm0", -- From arch definition
      ADL_Root & "/arch/ARM/Nordic/devices/nrf51", -- From MCU definition
      ADL_Root & "/arch/ARM/Nordic/drivers/nrf_common", -- From MCU definition
      ADL_Root & "/arch/ARM/Nordic/drivers/nrf51", -- From MCU definition
      ADL_Root & "/arch/ARM/Nordic/svd/nrf51/", -- From MCU definition
      ADL_Root & "/middleware/src/filesystem", -- From middleware config
      ADL_Root & "/middleware/src/BLE", -- From middleware config
      ADL_Root & "/middleware/src/utils", -- From middleware config
      ADL_Root & "/middleware/src/audio", -- From middleware config
      ADL_Root & "/middleware/src/monitor", -- From middleware config
      ADL_Root & "/middleware/src/bitmap", -- From middleware config
      ADL_Root & "/middleware/src/command_line", -- From middleware config
      ADL_Root & "/middleware/src/sdmmc", -- From middleware config
      ADL_Root & "/middleware/src/neopixel", -- From middleware config
```

```
        ADL_Root & "/components/src/**" -- From components config
      );
...
```

Gracias a utilizar como directorio de recursos la propia librería de AdaCore, se podrá utilizar en ejemplos propios, códigos pertenecientes a esta librería como el envío de datos a través del bus I2C.

3.5. Programación del timer para producir interrupciones

Ahora que se tiene M2OS instalado de forma correcta y, además, se puede compilar programas que corran sobre M2OS, es hora de realizar la programación del temporizador hardware (timer), ya que, una de las cosas más importantes del proyecto, es la ejecución de tareas y, para ello, necesitamos de un timer funcional.

M2OS es un sistema operativo basado en ticks, esto es que, cada cierto tiempo, un timer produce una interrupción. Un tick de un sistema operativo es la unidad de tiempo en la que se basan los timers de cualquier sistema. Este tick es un evento programado, es decir, puede producir una interrupción.

Las interrupciones en un sistema operativo se suelen programar con un periodo de tiempo que comprende desde 1 ms hasta 100 ms, pero puede ser más largo o más corto. El overhead que obtenemos con las interrupciones es cada vez más significativo cuanto más corto es el período, por lo que existe un equilibrio entre la resolución del temporizador y la sobrecarga de la CPU.

Una función handler o manejadora es una función utilizada por el sistema operativo para gestionar una interrupción. Cada vez que se produzca una determinada interrupción identificada con un nombre, se ejecutará su función manejadora la cual realizará una tarea predeterminada y, eliminará el aviso de interrupción para cuando se produzca otra.

Para la programación del timer se descomentó del fichero 'm2-hal.adb' la llamada a la función 'initialize_board' y se procedió a su implementación. Esta llamada se comentó al comienzo del proyecto para que la instalación del sistema operativo se realizase de forma correcta, sin embargo, ahora se necesita descomentar dicha llamada ya que, esta será la encargada de inicializar el timer.

Para la programación del timer se tomará como referencia como realiza la inicialización del timer en la librería 'Ada Drivers Library'. Cabe comentar que el micro:bit posee dos timers de tiempo real, el Real Time Counter 0 (RTC0) y el RTC1. La librería 'ADL' utiliza el RTC1 para producir las interrupciones del timer, mientras en M2OS utilizaremos el RTC0. De esta forma, podremos utilizar las tareas de M2OS a la vez que realizamos llamadas a la librería de Microbit.

Para algunos ejemplos integrados en la librería 'ADL', es necesario generar una onda sinusoidal. Para ello, 'ADL' hace uso de un timer, especificamente, el RTC1 (En M2OS utilizamos el RTC0), por lo que, para los ejemplos que utilicen esta técnica, será necesario modificar el nombre con el que se genera la interrupción del RTC1 a adl irq handler.

Esta modificación se realizó en el fichero

```
$ 'gnat_rts/microbit/rts-m2os-files/arm-eabi__BSPs__cortex-m__micro
bit__m2os__arch__m2os_specific_gnat_src/handler.S'.
```

```
...

.word __gnat_irq_trap        /* 21 IRQ 5. */
    .word __gnat_irq_trap        /* 22 GPIOTE */
    .word __gnat_irq_trap        /* 23 ADC */
    .word __gnat_irq_trap        /* 24 TIMER0 */
    .word __adl_irq_handler      /* 24 TIMER0 used by ADL*/
    .word __gnat_irq_trap        /* 25 TIMER1 */
    .word __gnat_irq_trap        /* 26 TIMER2 */
    .word __gnat_irq_trap_rtc0   /* 27 RTC0 used by M2OS*/

    ...

                .thumb_func
    .weak __adl_irq_handler
    .type __adl_irq_handler, %function
    __adl_irq_handler:
    0:        b 0b
            .size __adl_irq_handler, . - __adl_irq_handler
```

Código 1: Cambio de nombre interrupción RTC1 en handler.S

Utilizar dos timers es factible ya que cada interrupción de los timers se exportará con un nombre diferente. Por ejemplo, las interrupciones del RTC1 las podemos exportar con el nombre __adl_interrupt y las del RT0 como __M2OS_interrupt. De esta forma, cada interrupción va a tener su propia función handler para tratar las interrupciones cuando se produzcan.

Una vez se entiende que existen diferentes timers en microbit, fue necesario comprobar como se realiza la inicialización del RTC1 en la 'ADL'. Para ello, se tomará como referencia el paquete Microbit.Time donde, en la función Inicialize, se realiza esta inicialización.

```
procedure Initialize is
begin
 -- Si el reloj no esta en marcha, se coloca a cierta frecuencia y se inicia.
   if not Clocks.Low_Freq_Running then
      Clocks.Set_Low_Freq_Source (Clocks.LFCLK_SYNTH);
      Clocks.Start_Low_Freq;

 -- Se espera a que se inicialice el timer
      loop
         exit when Clocks.Low_Freq_Running;
      end loop;
   end if;

 -- Detener el timer
   Stop (RTC_1);

   --  1kHz
   Set_Prescaler (RTC_1, 0);
   Set_Compare (RTC_1, 0, 32);

   Enable_Event (RTC_1, Compare_0_Event);

   nRF.Events.Enable_Interrupt (nRF.Events.RTC_1_COMPARE_0);
```

```
-- Asignar el manejador correspondiente para las interrupciones del timer
      nRF.Interrupts.Register (nRF.Interrupts.RTC1_Interrupt,
                               RTC1_IRQHandler'Access);

    -- Habilitar las interrupciones del timer RTC1
      nRF.Interrupts.Enable (nRF.Interrupts.RTC1_Interrupt);
    -- Comenzar el timer
      Start (RTC_1);
    end Initialize;
```

Para hacer el 'portado' de esta inicialización del timer se debe de comprobar la instrucción específica que realiza cada una de las instrucciones que vemos en el código anterior y, una vez localizada la instrucción en concreto trasladar dicha instrucción a la inicialización de nuestro timer RTC0 en la función 'Initialize_Board' de nuestro m2-hal.adb.

Pongamos algunos ejemplos:

- La instrucción Stop(TIMER), si se navega a través de ella, la instrucción específica que ejecuta es 'This.Periph.TASKS_STOP := 1;'. Esta instrucción será la que se copiará a nuestro hal.
- La instrucción Set_Preescaler(TIMER, 0) ejecuta la instrucción 'This.Periph.PRESCALER.PRESCALER := Prescaler' donde Preescaler está compuesto por un tipo de dato Real_Time_Counter y un Uint12.

Si se traslada esta inicialización junto con las instrucciones que componen el manejador de las interrupciones nos queda algo parecido a esto en nuestro m2-hal.adb:

```
procedure Initialize_Board is
   --Declaramos todo lo necesario para inicializar el timer.
   use Interfaces.NRF51.CLOCK, Interfaces.NRF51.RTC;
   use type Interfaces.Nrf51.UInt12;
   use type Interfaces.NRF51.UInt32;
   use type Interfaces.NRF51.UInt24;

   Reg_Addr : constant UInt32 := To_UInt32 (System.Address
   (RTC0_Periph.EVENTS_COMPARE (0)'Address));
   Device_Base : constant UInt32 := Reg_Addr and 16#FFFF_F000#;
   Event_Index : constant UInt7 := UInt7 (Reg_Addr and 16#0000_007F#) / 4;
   Set_Register_Addr : constant UInt32 := Device_Base + 16#0000_0304#;
   Set_Register : UInt32 with Address => To_Address (Set_Register_Addr);

   NVIC_Base : constant System.Address := System'To_Address (16#E000E100#);
   type NVIC_Peripheral is record
   --  Interrupt Set-Enable Registers
   NVIC_ISER : aliased HAL.UInt32;
   end record
   with Volatile;

   NVIC_Periph : aliased NVIC_Peripheral
   with Import, Address => NVIC_Base;

   Value : constant Interfaces.NRF51.UInt24 := 33;

begin
   --  Mask interrupts
   Disable_Interrupts;
   -- Realizamos todas las instrucciones de la inicialización de ADL.
   --  Configure the low frequency clock required for RTC0
   CLOCK_Periph.LFCLKSRC.SRC := Rc; -- Use internal RC oscillator
   --  Start the low frequency clock
   CLOCK_Periph.TASKS_LFCLKSTART := 1;
   --  Wait until the low frequency clock is started
   while CLOCK_Periph.EVENTS_LFCLKSTARTED = 0 loop
      null;
   end loop;
   -- Stop timer
   RTC0_Periph.TASKS_STOP := 1;
   --  We run the counter at 32.768KHz
   RTC0_Periph.PRESCALER.PRESCALER := 0;
   RTC0_Periph.CC (Integer (0)).COMPARE := Value;
   -- Enable Events
   RTC0_Periph.EVTEN.COMPARE.Arr (0) := Enabled;
   -- Enable Interrupt
   Set_Register := 2**Natural (Event_Index);
   NVIC_Periph.NVIC_ISER := Shift_Left (1, Natural (11));
   --  Clear pending timer interrupt if any
   RTC0_Periph.EVENTS_TICK := 0;
   --  Now that the interrupt handler is attached, we can start the timer
   RTC0_Periph.TASKS_START := 1;
end Initialize_Board;
```

El código del handler se puede leer en el siguiente código:

```
   procedure Sys_Tick_Handler;
      pragma Export (C, Sys_Tick_Handler, "__gnat_irq_trap_rtc0");

      procedure Sys_Tick_Handler is
         use Interfaces.NRF51.CLOCK, Interfaces.NRF51.RTC;
         Reg : UInt32 with Address => System.Address (RTC0_Periph.EVENTS_COMPARE (0)'Address);
      begin

      -- IMPORTANTE ELIMINAR EL AVISO DE INTERRUPCION PARA QUE SE PUEDA
      -- VOLVER A PRODUCIR
         RTC0_Periph.TASKS_STOP := 1;
         RTC0_Periph.TASKS_CLEAR := 1;
         RTC0_Periph.TASKS_START := 1;

         Reg := 0;

         Now := Now + 1;

         if M2.Use_Timing_Events'First then
            OS_Tick_Handler.all;
         end if;
   end Sys_Tick_Handler;
```

Es muy importante 'portar' también el handler ya que, este es el encargado de eliminar el aviso de interrupción, por lo que, en el caso de que no se realice el CLEAR de la interrupción, no se producirá más que una única interrupción.

Una vez portado el código del timer, se probará el ejemplo periodic_task del proyecto de ejemplos básicos de micro:bit (donde estaba el hello_world.adb) para ello, se utilizará el siguiente comando:

```
& gnatstudio -P examples/api_m2os/examples_api_m2os_microbit.gpr &
```

Una vez abierto se cargará el ejemplo periodic_task.adb en nuestro microcontrolador. Si se obtiene la siguiente salida por consola, significa que M2OS es capaz de hacer que las tareas se suspendan y se activen gracias a la correcta configuración del timer.

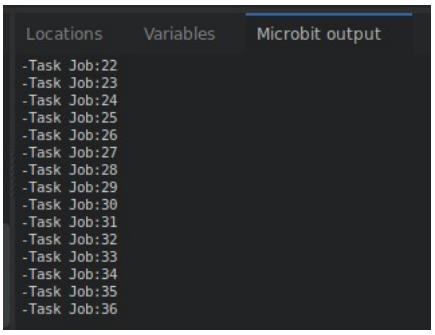

Figura 13: Salida del programa periodic_task.adb

3.6. Configuración del timer a la frecuencia correcta

Una vez se tiene el timer funcional, se debe de configurar con la velocidad de ticks correcta, es decir, que produzca interrupciones a una velocidad lo suficientemente rápida como para que tenga precisión pero no tan rápido como para que produzcamos un overhead en el procesador del microcontrolador.

Para ello, se fijará para que produzca una interrupción cada 1ms. Para ello, se modificó en los ficheros m2-hal.ads y en el fichero s-osinte.ads la variable HWtime a 1024, la variable preescaler lo pondremos a 0 ya que es así como lo configuran en la librería 'ADL' y el counter lo pondremos a valor 32.

De acuerdo a la ecuación FreqReloj = 32768Hz / (preescaler + 1), con los datos introducidos tenemos que FreqReloj=32768/(0+1)=32768Hz. Esta frecuencia quiere decir que cada tick de reloj se produce cada 1/FreqReloj=1/32768Hz=0,000030518s. Al establecer el counter a 32 se produce una interrupción cada 0,000030518s×32=0,000976562s.

Sin embargo, este valor no es lo suficientemente exacto ya que en un largo periodo de tiempo se produce una desviación de tiempo considerable. Para acercarnos más a un valor más preciso, colocaremos el valor de HWtime de los ficheros m2-hal.ads y s-osinte.ads a 1000, preescaler lo dejaremos a 0 y el counter lo situaremos a 33.

Con esta configuración de timer obtenemos una interrupción cada 0,001007094s siendo este un valor más preciso que el anterior.

4. Otras aportaciones.

4.1. Incorporación de salida del puerto serie en el IDE de GNAT

Otro aspecto importante en el desarrollo de este proyecto, fue la programación de alguna forma de visualización de la salida por consola de micro:bit dentro del propio IDE de GNAT.

AdaCore pone a disposición de los usuarios una guía con todas las características, funciones y propiedades que ofrece el IDE de GNAT. Una de estas características trata sobre la modificación de la interfaz del propio IDE mediante ficheros .xml y .py.[14]

El funcionamiento es muy sencillo, GNAT Studio busca scripts de tipo .xml o .py cuando se inicia en varios directorios diferentes. Dependiendo de dónde se encuentren estos ficheros, GNAT Studio los carga automáticamente. De entre todos los directorios, usaremos el directorio INSTALL/share/gnatstudio/plug-ins (Donde INSTALL es el directorio de instalación de GNAT y es donde se cargan scripts por defecto).

Una vez que hemos decidido donde vamos a colocar el script, es necesario programar dicho script. En mi caso, lo programaré en .xml y seguiré la guía de AdaCore. El resultado final del script se muestra en el código 1.

```
<test>
    <button action="USB/Serial console for Microbit" iconname="gps-paste-symbolic"/>
    <action name="USB/Serial console for Microbit" show-command="false"
    output="Microbit output">
        <external>echo -n Microbit output enabled: Waiting to flash the board..</external>
        <external>printf "\n\n"</external>
        <external>stty -F /dev/ttyACM0 cs8 115200 ignbrk -brkint -icrnl -imaxbel -opost
        -onlcr -isig -icanon -iexten -echo -echoe -echok -echoctl -echoke noflsh -ixon
        -crtscts</external>
        <external>cat /dev/ttyACM0</external>
    </action>
</test>
```

Código 2: Script para añadir un botón a GNAT

El script es muy sencillo, comenzamos creando un nuevo botón en el IDE de GNAT con el icono 'gps-paste-symbolic' que, cuando se pulse, ejecutará la acción denominada 'USB/Serial console for Microbit', la cual nos creará una ventana nueva denominada 'Microbit output'. Esta acción tiene marcada la opción show-command a 'false' para que no nos salga en la nueva ventana abierta los comandos que se van a ejecutar a continuación con la opción 'external' la cual nos indica que van a ejecutarse comandos de la shell.

Los comandos que se ejecutan al pulsar el botón son:

- Al pulsar el nuevo botón añadido y haberse abierto la ventana, se escribirá (meramente estético) la frase 'Microbit output enabled: Waiting to flash the board.' junto con dos saltos de linea. Esto hará que el usuario al darle al botón, sepa que tiene que darle al botón de flashear la placa para ver algo por esa nueva ventana.
- Cuando se pulsa el botón, a parte de la frase escrita para informar al usuario, es necesario ejecutar también el comando que utilizamos en una primera instancia para visualizar por una terminal la salida de nuestro micro:bit. De esta forma, cuando el usuario ejecute un programa en la placa, la salida podrá verse en esta nueva ventana de GNAT.

Este script tiene una desventaja, siempre que abra de nuevo el IDE de GNAT tendrá que volver a darle al nuevo botón para que se le habrá la nueva ventana y poder visualizar la salida del micro:bit.

4.2. Incorporación del script a instalación de M2OS

Una vez que ya tenemos listo el script, era necesario incorporar la instalación de este en la instalación de M20S con el principal objetivo de que el botón únicamente apareciese cuando la arquitectura instalada de M2OS es la de micro:bit.

Para realizar esta adición fue necesario modificar varios ficheros:

- Lo primero que hacer fue guardar el script en la carpeta scripts/gps de M2OS, donde ya se encontraban dos scripts utilizados en Arduino. El nuevo script tomó el nombre de 'microbit_examples.xml'.
- Lo segundo fue modificar el Makefile de M2OS para que en el caso de que la arquitectura a instalar fuese la de microbit, copiase el script 'microbit_examples.xml' a la carpeta de plugins de GNAT:

```
# Copy scripts to automatize Arduino or Microbit development in GPS or GNAT
copy_gps_scripts:
ifeq ($(M2_TARGET), arduino_uno)
        @echo "-- Copying GPS scripts to .gps/plug-ins ..."
        mkdir -p ~/.gps/plug-ins/
        ln -s -f $(M2OS)/scripts/gps/m2_buttons_arduino_uno.py ~/.gps/plug-ins/
        ln -s -f $(M2OS)/scripts/gps/m2_build_actions_arduino_uno.py ~/.gps/plug-ins/
else if ($(M2_TARGET), microbit)
        @echo "-- Copying GPS script to .gnatstudio/plug-ins ..."
        mkdir -p ~/.gnatstudio/plug-ins/
        ln -s -f $(M2OS)/scripts/gps/examples_microbit.xml ~/.gnatstudio/plug-ins/
endif
```

Código 3: Adición de else if en el Makefile de M2OS

  Este condicional significa que, si la arquitectura a instalar es microbit, se procederá a crear el directorio de plugins en el directorio que la necesita GNAT (si es que no existe) y, en ese caso, generar un enlace simbólico del script situado en el directorio de instalación de M2OS al directorio de GNAT.

  De esta forma, cuando se utilice GNAT Studio con M2OS y sobre micro:bit, aparecerá este nuevo botón.

- Para finalizar con esta implementación del script, también fue necesario añadir al fichero rules.mk, que cuando se desinstale M2OS, también se borren los scripts que han sido copiados.

```
clean: clean_gprs clean_dirs clean_global clean_scripts

... (Código no importante)

clean_scripts:
    @exec echo -e "\n>> Scripts cleaning (Arduino_uno, Microbit,...)... ";
    rm -f ~/.gps/plug-ins/m2_buttons_arduino_uno.py
    rm -f ~/.gps/plug-ins/m2_build_actions_arduino_uno.py
    rm -f ~/.gnatstudio/plug-ins/examples_microbit.xml
    @exec echo ">> End Cleaning M2OS Scripts"
```

  Con esta adición, forzamos a que todos los scripts sean eliminados de sus directorios cuando se ejecuta un 'make clean' de M2OS.

5. Evaluación y pruebas.

5.1. Compilar primer programa en M2OS

La primera prueba que se realizó en el proyecto tuvo lugar una vez tenemos la instalación de la arquitectura, esta prueba trataba de ejecutar un pequeño programa de prueba sobre M2OS corriendo sobre nuestra placa micro:bit.

Una vez estamos en condiciones de compilar y correr nuestro primer programa en M2OS sobre micro:bit, abriremos el proyecto de ejemplos de M2OS localizado en la ruta 'examples/api_m2os/examples_api_m2os_microbit.gpr'.

Este proyecto contiene dos ejemplos básicos para ejecutar en M2OS independientemente de la placa sobre la que estemos trabajando. Estos dos ejemplos son:

- hello_world: Es el ejemplo más básico, consiste únicamente en escribir por consola utilizando el paquete Ada.Text_IO.
- periodic_task: Es un ejemplo básico que implementa tareas en M2OS. Este ejemplo no funcionará de momento ya que al tratarse de tareas necesita de un timer funcional.

Para abrir el proyecto de ejemplos simplemente deberemos de utilizar el siguiente comando:

```
$ gnatstudio -P examples/api_m2os/examples_api_m2os_microbit.gpr &
```

Al abrir el proyecto, si mantenemos el botón de click izquierdo sobre el botón de cargar en la placa, podremos seleccionar que ejemplo queremos compilar y cargar. Antes de realizar la carga sobre la placa, tenemos que acordarnos de abrir una terminal y ejecutar el comando para visualizar la salida del dispositivo como ya mencionamos previamente.

Una vez ejecutado el comando en una terminal y cargado el ejemplo hello_world en la placa, deberíamos obtener la siguiente salida por la terminal:

```
mario@MrBeast:~/Documentos/TFG/M2OS-microbit$ stty -F /dev/ttyACM0 cs8 115200 ignbrk -brkin
t -icrnl -imaxbel -opost -onlcr -isig -icanon -iexten -echo -echoe -echok -echoctl -echoke
noflsh -ixon -crtscts && cat /dev/ttyACM0
-=M2OS=-
M2.Direct_IO: Hello world
Text_IO: Hello world
GNAT.IO: Hello world
```

Figura 14: Salida del programa hello_world.adb

Esto significa que tenemos M2OS correctamente portado y corriendo sobre el microcontrolador micro:bit.

5.2. Ejecución de los ejemplos de ADL sobre M2OS

Una vez que ya pudimos compilar un primer programa en M2OS y comprobado el correcto funcionamiento del timer en M2OS, se procedió a ejecutar una serie de ejemplos que nos permitieran verificar el correcto funcionamiento de la inicialización de pines, señales analógicas, digitales, acelerómetro etc...

Para ello, 'Ada Drivers Library' incluye en su librería una serie de ejemplos para ejecutar sobre micro:bit. Como estos ejemplos ya están programados por AdaCore, hicimos una migración de estos para ejecutarles sobre M2OS.

Para la prueba de los ejemplos se utilizaron algunos elementos adicionales como:

- Una placa de pruebas (breadboard): Se trata de un tablero con orificios que se encuentran conectados eléctricamente entre sí de manera interna, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos, cables para el armado, prototipado de circuitos electrónicos y sistemas similares.
- Un potenciómetro: Es un elemento que al manipularlo, obtiene entre el terminal central y uno de los extremos una fracción de la diferencia de potencial total, se comporta como un divisor de tensión pudiendo ajustar, por ejemplo, la intensidad de una luz.
- Un par de resistencias eléctricas.
- Un servomotor: Es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición.
- Un zumbador: Produce un zumbido continuo o intermitente de un mismo tono (generalmente agudo). Sirve como mecanismo de señalización o aviso y se utiliza en múltiples sistemas, como en automóviles o en electrodomésticos, incluidos los despertadores.

Algunos de los ejemplos utilizados y el proceso de testeo se detalla a continuación:

5.2.1. Botones

El objetivo de este ejemplo es comprobar el correcto funcionamiento de los dos botones que trae integrado el micro:bit.

El ejemplo básicamente imprime por el display de micro:bit el símbolo '¡' si se pulsa el botón de la izquierda o el símbolo '¿' si se pulsa el botón de la derecha.

```
procedure Buttons is
begin
   Ada.Text_IO.Put_Line ("Buttons working in M2OS!");
   loop
      MicroBit.Display.Clear;

      if MicroBit.Buttons.State (Button_A) = Pressed then
         --  If button A is pressed

         --  Display the letter A
         MicroBit.Display.Display ('¿');
         Ada.Text_IO.Put_Line ("Se pulsa el boton de la izquierda!");
         MicroBit.IOs.Set (12, True);
      elsif MicroBit.Buttons.State (Button_B) = Pressed then
         --  If button B is pressed

         --  Display the letter B
         MicroBit.Display.Display ('!');
         Ada.Text_IO.Put_Line ("Se pulsa el boton de la derecha!");
      else
         MicroBit.IOs.Set (8, False);
         MicroBit.IOs.Set (12, False);
      end if;

      MicroBit.Time.Delay_Ms (200);
   end loop;
end Buttons;
```

5.2.2. Zumbador

Otra prueba realizada fue la del zumbador, esta prueba constaba de intentar reproducir alguna canción o sonido por el zumbador que incorpora el micro:bit. Para este ejemplo, se realizó el siguiente montaje en la breedboard:

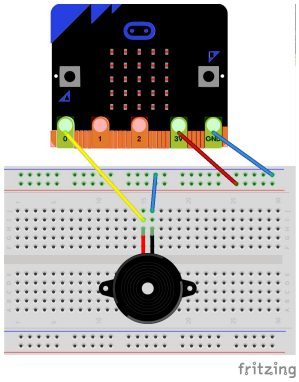

Figura 15: Instalación del zumbador sobre Breedboard.

La instalación es sencilla, simplemente debemos alimentar el zumbador conectando el pin de 3V y el pin de tierra (GND) y por el pin 0 iremos mandando diferentes valores de tensión para que se produzca un sonido por el zumbador.

Este ejemplo, junto con dos que veremos posteriormente, utilizan el PWD (Modulación por ancho de pulso). Se trata de una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

Para realizar la prueba se creará una constante de tipo .Melody especificada por AdaCore (en mi caso usaré la canción de Piratas del Caribe) y después utilizaremos la instrucción Microbit.Music.Play donde el primer argumento es el pin por donde se hará la variación de tensión y el segundo argumento la canción o sonido a reproducir.

```
...

procedure MusicM2 is

   Pirates_of_the_Caribbean : constant MicroBit.Music.Melody :=
     ((E4,   125), (G4, 125), (A4,   250), (A4, 125), (Rest, 125),
      (A4,   125), (B4,   125), (C5,   250), (C5,   125), (Rest, 125),
      (C5,   125), (D5,   125), (B4,   250), (B4,   125), (Rest, 125),
      (A4,   125), (G4,   125), (A4,   375), (Rest, 125),

    ...

     );
begin
   Ada.Text_IO.Put_Line ("Music working in M2OS!");
   --  Loop forever
   loop
      --  Play Pirates of the Caribbean on pin 0
      MicroBit.Music.Play (0, Pirates_of_the_Caribbean);
   end loop;
end MusicM2;
```

5.2.3. Entrada y Salida Analógica

Para este ejemplo utilizamos una luz, una resistencia, un potenciómetro y nuestro micro:bit:

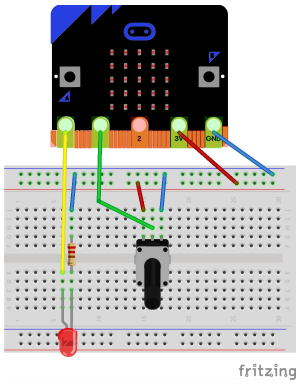

Figura 16: Instalación del potenciómetro sobre Breedboard.

El ejemplo básicamente consiste en regular la luminosidad de la bombilla en función de la posición del potenciómetro, ya que, dependiendo de su posición, a la bombilla le llegará más o menos tensión.

Para la entrada/salida analógica tenemos dos ejemplos diferentes. El ejemplo de analog_in es un ejemplo sencillo que utiliza el Microbit.IOs.Analog(PIN) para obtener el valor de dicho pin y, una vez obtenido se escribe sobre el pin de la bombilla para variar su valor.

```
with MicroBit.IOs;
with Ada.Text_IO;

with AdaX_Dispatching_Stack_Sharing;

procedure Analog_In is

   Value : MicroBit.IOs.Analog_Value;
   Old_Value : MicroBit.IOs.Analog_Value;
begin
   Ada.Text_IO.Put_Line ("Analog In working in M2OS!");
   --  Loop forever
   loop

      --  Read analog value of pin
      Value := MicroBit.IOs.Analog (1);
      if(Value'Image /= Old_Value'Image) then
         Ada.Text_IO.Put("Sensor Value: ");
         Ada.Text_IO.Put_Line(Value'Img);
         Old_Value := Value;
      end if;

      --  Write analog value of pin 0
      MicroBit.IOs.Write (0, Value);

   end loop;
end Analog_In;
```

Por otro lado, el ejemplo analog_out.adb consiste en un bucle que le da a la bombilla un valor entre 0 y 128 de forma cíclica. En resumen, el ejemplo analog_in.adb utiliza de forma interna el ejemplo analog_out.adb.

```
procedure Analog_Out is
begin
   Ada.Text_IO.Put_Line ("Analog Out working in M2OS!");
   --  Loop forever
   loop

      --  Loop for value between 0 and 128
      for Value in MicroBit.IOs.Analog_Value range 0 .. 128 loop

         --  Write the analog value of pin 0
         MicroBit.IOs.Write (8, Value);

         --  Wait 20 milliseconds
         MicroBit.Time.Delay_Ms (20);
      end loop;
   end loop;
end Analog_Out;
```

5.2.4. Entrada y Salida Digital

Estos dos ejemplos son muy similares entre si. El principal objetivo de estos ejemplos es realizar entrada/salida digital a traves de los pines de nuestro micro:bit.

Ambos ejemplos utilizan la instruccion Microbit.IOs.Set(PIN, True o False) para enviar un 0 o un 1 por el pin que queramos. Asimismo, podemos utilizar ese mismo set en un condicional If para comprobar si se esta enviando un 0 o un 1 por un pin determinado.

A continuación muestro el código del ejemplo digital_in.adb:

```
procedure Digital_In is
begin
   Ada.Text_IO.Put_Line ("Digital In working in M2OS!");
   --  Loop forever
   loop

      --  Check if pin 1 is high
      if MicroBit.IOs.Set (1) then

         --  Turn off the LED connected to pin 0
         Ada.Text_IO.Put_Line ("Luz Apagada");
         MicroBit.IOs.Set (0, False);
      else

         --  Turn on the LED connected to pin 0
         Ada.Text_IO.Put_Line ("Luz Encendida");
         MicroBit.IOs.Set (0, True);
      end if;
   end loop;
end Digital_In;
```

5.2.5. Beacon Bluetooth

Este ejemplo es bastante más complejo, el principal objetivo de este ejemplo es generar un beacon bluetooth con el micro:bit. De esta forma, micro:bit será visible para otros dispositivos bluetooth como nuestro

teléfono móvil.

Para comprobar el correcto funcionamiento de este ejemplo en M2OS, es necesario bajarse una app en nuestro smartphone que detecte beacons. Si con nuestro smartphone captamos el beacon de nuestro micro:bit, significará que somos capaces de generar beacons bluetooth correctamente con micro:bit sobre M2OS.

5.3. Ejemplos con Maqueen

Una vez hemos comprobado con los ejemplos de AdaCore corriendo sobre M2OS las principales utilidades del microcontrolador, podemos pasar a probar estos ejemplos con el micro:bit conectado con en el robot Maqueen, con el objetivo de comprobar que los principales componentes del Maqueen como su zumbador, sus luces, sus sensores etc.. funcionan de forma correcta.

5.3.1. Modificación de ejemplos para funcionar con Maqueen

Para que los ejemplos que hemos probado previamente sobre micro:bit funcionen de forma correcta sobre Maqueen, es necesario modificar únicamente los pines sobre los que queremos actuar, es decir, si previamente para encender una luz en el breadboard hemos utilizado el pin 3, será necesario cambiar ese pin al correspondiente a la luz del Maqueen.

Pongamos un ejemplo, una de las luces LED frontales del Maqueen corresponde al pin número 8. Si ejecutásemos el código visto previamente de digital_out.adb con el micro:bit conectado al Maqueen y cambiando en la función 'MicroBit.IOs.Set (0, True)' el pin 0 por un 8, encenderíamos la luz izquierda de nuestro robot.

De esta forma se fueron modificando los ejemplos vistos previamente (Sobretodo los analog in/out y los digital in/out) para comprobar que todos los elementos del Maqueen funcionaban de forma correcta.

5.3.2. Motores del Maqueen por I2C

Ahora que hemos probado las principales entradas y salidas por los pines del micro:bit y viendo que somos capaces de interactuar con el robot Maqueen, un aspecto clave era hacer que funcionasen los dos motores que el Maqueen tiene integrado.

La comunicación con los motores se realiza a través del protocolo I2C que fue explicado previamente. Gracias a que utilizamos recursos de la librería de AdaCore no fue necesario la programación del protocolo, simplemente deberemos de mandar los datos a la dirección correcta y en el orden correcto. Para ello, utilizaremos la función Master_Transmit que nos proporciona AdaCore.

Para saber que datos hay que enviar, el orden en el que tenemos que enviarlos y la dirección a la que hay que enviarlos, nos basaremos en la librería de Maqueen para Arduino[7].

Si nos dirigimos al fichero src/Maqueen.cpp veremos el código de algunas funciones, la que a nosotros nos interesa es cualquiera que comunique algo a los motores como MotorStop o MotorRun, el siguiente código muestra la función motorRun:

```
void Maqueen::motorRun(int motor, int direction, int speed) {
  Wire.beginTransmission(0x10);
  Wire.write((byte)motor);
  Wire.write((byte)direction);
  Wire.write((byte)speed); //speed
```

```
  Wire.endTransmission();
}
```

Como vemos, la dirección correspondiente a los motores es la dirección 0x10, A esta dirección deberemos de mandar como primer byte, el motor que queramos mover (0x00 o 0x01), posteriormente un byte indicando la dirección adelante/atrás (0x00 o 0x01) y, finalmente, la velocidad a la que lo queremos mover (Valor comprendido entre 0 y 250). Sabiendo esta información, se procedió a crear un ejemplo utilizando la librería de AdaCore para realizar la misma comunicación con nuestro micro:bit. El ejemplo generado fue el siguiente:

```
procedure Forward
   is
      Ctrl   : constant Any_I2C_Port := MicroBit.I2C.Controller;
      Addr   : constant I2C_Address := 16#10#;
      Data   : I2C_Data (0 .. 2);
      Status : I2C_Status;
   begin
      if not MicroBit.I2C.Initialized then
         -- I2C is not initialized
         Raise Program_Error with "Motors has not been initialized!";
      end if;

      Data := (0, 0, Hal.Uint8(MotorsSpeed));
      Ctrl.Master_Transmit(Addr => Addr, Data => Data, Status =>Status);
      Data := (16#2#, 0, Hal.Uint8(MotorsSpeed));
      Ctrl.Master_Transmit(Addr => Addr, Data => Data, Status =>Status);
   end Forward;
```

El ejemplo es muy sencillo, comenzamos inicializando una serie de variables que necesita el transmit del I2C para funcionar. El puerto I2C el cual lo obtenemos con la variable MicroBit.I2C.Controller, la dirección del dispositivo (0x10 en el ejemplo) un array con los bytes a enviar de 3 elementos y una variable de tipo status para el retorno del transmit. Después simplemente rellenamos el array con los bytes a enviar y realizaremos el transmit a la dirección correspondiente.

Sin embargo, el retorno de la función Master_Transmit que obteníamos era de Error. Tras investigar un poco, vimos que el error era debido a múltiples factores:

- Lo primero que vimos fue que a la hora de mandar la dirección por el bus I2C, el protocolo programado por AdaCore nos elimina el último bit, por lo que, en este, caso, cuando intentamos mandar un 0x10 cuyo valor en binario es 10000, en realidad, estábamos mandando la dirección 0x08. Para evitar esto, la dirección que deberemos de mandar será el doble, es decir, 0x20, de esta forma, cuando utilicemos la función de Master_Transmit, se enviará la dirección 0x10.

- Otro problema que nos encontramos es que al mandar la dirección 0x20 tampoco conseguimos hacer funcionar los motores del Maqueen. La función nos seguía retornando en su variable Status un Error. Lo siguiente que decidimos probar es hacer un 'probe' para ver las direcciones disponibles en el bus I2C de nuestro micro:bit conectado al robot Maqueen. Para realizar este análisis del bus utilizamos un código perteneciente a la librería Baremetal[8]. Específicamente el ejemplo x18-level/ nos ofrece un análisis del bus I2C. Para ejecutarlo nos descargaremos los ficheros, los compilaremos con su correspondiente Makefile y lo cargaremos en nuestro micro:bit arrastrando el fichero generado por el Makefile.

  Para ver la salida que nos produce el análisis del bus utilizaremos el programa CuteCom configurado a 9600 baudios. La salida que obtenemos es la siguiente:

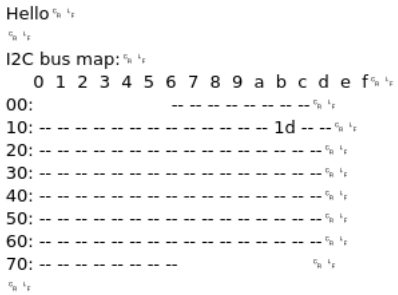

Figura 17: Salida del programa x18-level por CuteCom

Como apreciamos en la figura, solo tenemos disponible la dirección 0x1D que se corresponde con el acelerómetro del micro:bit con lo que, al intentar mandar cualquier byte a la dirección 0x20 obteníamos un error.

Investigando, se observó que era debido a la inicialización que realizada la propia librería de Ada-Core sobre los pines del I2C (SCA Y SCL) se realizaba con una configuración errónea. Leyendo el manual de la placa nRF51_RM_v3.0[15] la configuración necesaria para asegurar el correcto funcionamiento viene descrito en la página 145, tabla 258.

Al trasladar la siguiente configuración de pines a 'Ada Drivers Library' en la función de inicialización del bus I2C en el fichero boards/MicroBit/src/microbit-i2c.adb nos queda lo siguiente:

```
   procedure Initialize (S : Speed := S400kbps) is
     Config   : constant GPIO_Configuration := (Mode => Mode_In,
                                                Resistors => Pull_Up,
                                                Input_Buffer => Input_Buffer_Connect,
                                                Drive => Drive_S0D1,
                                                Sense => Sense_Disabled);

begin
   Device.Configure
     (SCL   => MB_SCL.Pin,
      SDA   => MB_SDA.Pin,
      Speed => (case S is
                   when S100kbps => nRF.TWI.TWI_100kbps,
                   when S250kbps => nRF.TWI.TWI_250kbps,
                   when S400kbps => nRF.TWI.TWI_400kbps)
     );

   --  Initialize the GPIO Pins for SCL & SDA with the
   -- configuration
   Configure_IO (MB_SCL, Config);
   Configure_IO (MB_SDA, Config);

   Device.Enable;
   Init_Done := True;
end Initialize;
```

Este código añadido simplemente crea una constante de tipo configuración de pines GPIO (Estructura de datos) y los inicializa como se indica en el manual de la placa nRF51_RM_v3.0[15]. Una vez declarada la constante, se aplicará dicha configuración para los pines SDA y SCL que componen el protocolo I2C. Tras aplicar esta configuración, si se ejecuta de nuevo el análisis del bus I2C obtenemos lo siguiente:

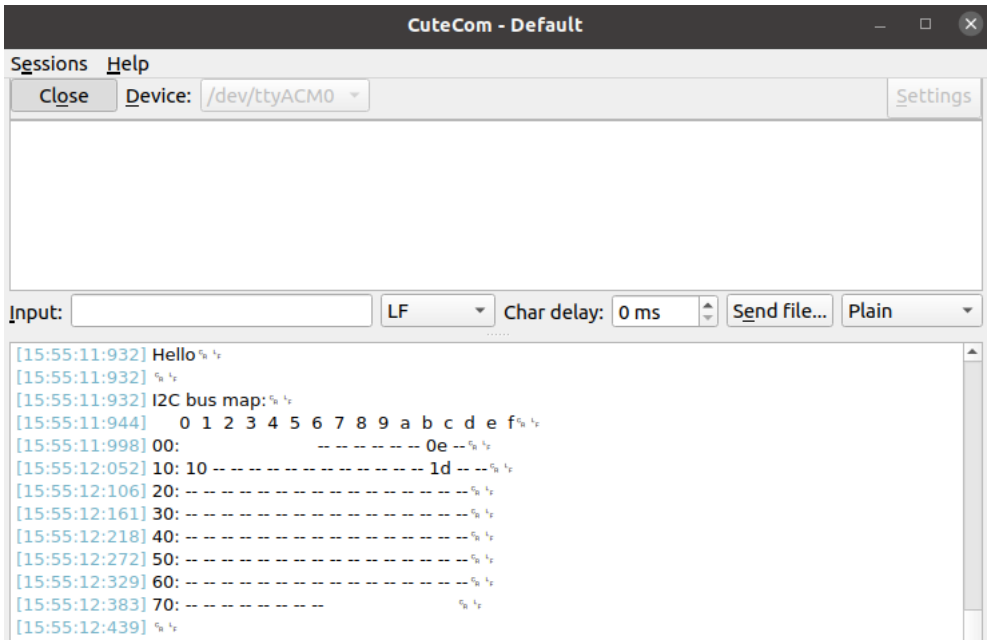

Figura 18: Salida del programa x18-level por CuteCom tras configuración

Como podemos ver, ahora sí que nos aparece la dirección 0x10 correspondiente a los motores del Maqueen, por lo que, si volvemos a ejecutar el ejemplo previo, conseguiremos que nuestro Maqueen se mueva hacia delante.

Dado que tuvimos que modificar algo de código perteneciente a 'ADL' se solicitó un branch merge para que esta solución formase parte del propio 'ADL'.

5.4. Manejador del sensor de proximidad sharp gp2d120

Con el objetivo de añadir más funcionalidades a nuestro robot Maqueen, se optó por añadir un sensor de distancia para que nuestro robot se detuviese al detectar un obstáculo. Este sensor de distancia es el sharp gp2d120, este sensor utiliza emisores de infrarrojos y foto detectores con circuitos de procesamiento de señal integrados. El dispositivo dispone de salidas analógicas o conmutadas (digitales) e incluyen sensores de distancia fijos y variables.

La salida del sensor se realiza a través de una señal analógica de valor proporcional a la distancia, por ejemplo, si un obstáculo se encuentra a una distancia de 20cm, si leemos el pin al que se encuentre conectado el medidor de distancia se obtendrá un valor de 600.

Figura 19: Sensor de distancia sharp gp2d120

5.4.1. Implementación

La instalación de este sensor es muy sencilla. Para su conexión necesitamos una alimentación de 3V que la sacaremos de los pines del propio Maqueen junto con la señal de tierra (GND), además, necesitaremos un pin para obtener el valor que nos retorna el sensor para saber la distancia del obstáculo, en mi caso,

elegí el pin 1.

Una vez conectado el sensor fue necesario realizar un pequeño programa de ejemplo que frenase al Maqueen en el caso de que se detectase un obstáculo a menos de escasos centímetros. El programa de prueba es el siguiente:

```
procedure Sensor is
      Value : MicroBit.IOs.Analog_Value := 0;
   begin

      --  Read analog value of pin 1
      Value := MicroBit.IOs.Analog (1);

      if not (Maqueen.Is_Running and Value < 600) then
         Maqueen.Forward;
      end if;

      if (Maqueen.Is_Running) then
         if (Value > 400) then
            M2.Direct_IO.Put_Line(Value'Img);
            Maqueen.Stop;
         end if;
      end if;
end Sensor;
```

El ejemplo es muy simple, básicamente consiste en leer de forma continua el pin 1 para obtener la distancia (En el código no hay ningún bucle debido a que el ejemplo se implementó como una tarea que se llamaba cada 250 ms). Si la distancia es menor de 600 (Valor correspondiente a una distancia de aproximadamente 20 cm) ordenaremos al Maqueen detenerse.

6. Programación de la librería Maqueen para micro:bit

6.1. Motivación

Como este proyecto está destinado en gran parte a los usuarios que quieran probar sus propios códigos utilizando M2OS sobre micro:bit y Maqueen, se decidió programar una librería con las funciones básicas que podemos hacer con dicho robot. De esta forma, los usuarios no necesitarán tener conocimientos de como funciona a bajo nivel la comunicación, por ejemplo, con los motores, si no que gracias a la librería simplificaremos esta comunicación a una única instrucción, por ejemplo, Maqueen.Fordward, la cual permitirá que el robot se desplace hacia delante.

Uno de los principales objetivos de esta librería es que cumpla con el formato estándar de los programas creados en Ada, es decir, que tenga su archivo de especificación .ads y su archivo de código .adb.

6.2. Especificación

Nuestra Libreria incluye las siguientes funcionalidades:

```
package Maqueen is
    -- Luces delanteras
   type Light is (Left_Light, Right_Light);
   -- Motores
   type Motor is (Left_Motor, Right_Motor);
   -- Direcciones
   type Direction is (Forwards, Backwards);
   -- Velocidad
   type Speed is range 0 .. 250;
   -- Encender la luz
   procedure Turn_On_Light (FrontLight : Light);
   -- Apagar la luz
   procedure Turn_Off_Light (FrontLight : Light);
   -- Reproducir Sonido
   procedure Play_Sound (Snd : Melody);
   -- Inicializar Robot
   procedure Initialize;
   -- Moverse hacia delante
   procedure Forward;
   -- Moverse hacia atrás
   procedure Backward;
   -- Establecer velocidad
   procedure SetSpeed(Spd : Speed);
   -- Rotar a la izquierda
   procedure Spin_Left;
   -- Rotar a la derecha
   procedure Spin_Right;
   -- Girar a la izquierda
   procedure Turn_Left;
   -- Girar a la derecha
   procedure Turn_Right;
   -- Deternerse
   procedure Stop;
   -- Mover motor independiente
   procedure Motor_Run(MotorToMove : Motor; WhichDirection : Direction; Spd : Speed);
   -- Saber si Maqueen se mueve
   function Is_Running return Boolean;
```

```
   -- Leer sigue lineas derecho
   function Read_Patrol_Right return Boolean;
   -- Leer sigue lineas izquierdo
   function Read_Patrol_Left return Boolean;
   -- Leer si la luz izq. está encendida
   function Left_Light_Status return Boolean;
   -- Leer si la luz dcha. está encendida
   function Right_Light_Status return Boolean;
end Maqueen;
```

6.3. Implementación

La implementación es una mezcla de todos los ejemplos previos y que, sobretodo utiliza funciones de Ada Drivers Library.

A continuación se detallan algunas de las funciones de las que se compone nuestra librería del Maqueen:

- Procedimiento para apagar luces (Mismo procedimiento para encender pero con valores True)

```
procedure Turn_Off_Light (FrontLight : Light)
is

begin
  case FrontLight is
    when Left_Light =>
       --  Turn on the LED connected to pin 8
        MicroBit.IOs.Set (8, False);

     when Right_Light =>
       --  Turn on the LED connected to pin 12
       MicroBit.IOs.Set (12, False);
    when others =>
       raise Program_Error;
  end case;
end Turn_Off_Light;
```

  Dependiendo del la luz que el usuario introduzca como parámetro (Izquierda o derecha) dicha luz se apagará.

- Procedimiento para indicar movimiento al Maqueen (Idénticos en todos los procedimientos, únicamente varían los bytes que transmitimos por I2C con el Master_Transmit)

```
procedure Forward
is
  Ctrl   : constant Any_I2C_Port := MicroBit.I2C.Controller;
  Addr   : constant I2C_Address := 16#20#;
  Data   : I2C_Data (0 .. 2);
  Status : I2C_Status;
begin
  if not MicroBit.I2C.Initialized then
     --  I2C is not initialized
     Raise Program_Error with "Motors has not been initialized!";
  end if;

  Data := (0, 0, Hal.Uint8(MotorsSpeed));
  Ctrl.Master_Transmit(Addr => Addr, Data => Data, Status =>Status);
```

```
    Data := (16#2#, 0, Hal.Uint8(MotorsSpeed));
    Ctrl.Master_Transmit(Addr => Addr, Data => Data, Status =>Status);

    Running := True;
  end Forward;
```

Dependiendo del procedimiento que se ejecute se mandaran unos bytes o otros a traves del bus I2C.

- Función que nos indica el estado del Maqueen (En movimiento o quieto)

```
function Is_Running return Boolean
is
begin
  return Running;
end Is_Running;
```

Running es un booleano declarado como variable global en el .adb que va siendo modificado por los procedimientos que pongan en marcha o detengan el Maqueen.

7. Demostrador.

7.1. Maqueen con cuatro tareas simultaneas

Para finalizar con el proyecto, se realizó un programa multitarea de mediana complejidad el cual sobrepasase el límite de tres tareas simultáneas debido a la memoria del microcontrolador. Este ejemplo final se programó haciendo uso de la librería programada en el apartado anterior para el robot Maqueen.

El ejemplo en cuestión consistirá en un robot "sigue líneas"que se detenga cuando detecte un obstáculo en su camino. La implementación consistirá en un programa compuesto por cuatro tareas. La funcionalidad de cada una de ellas se detalla a continuación:

- La primera tarea consistirá en controlar la luz izquierda. Esta tarea se ejecutará cada 2 segundos. En cada ejecución se comprobará el estado de dicha luz. Si la luz está apagada se encenderá y, si por el contrario, está encendida, se apagará.
- La segunda tarea consistirá en controla la luz derecha de la misma forma que la izquierda salvo que está tarea entrará a ejecutar cada segundo.
- La tercera tarea consistirá en comprobar el sensor de distancia cada 250ms para comprobar si delante del robot Maqueen existe un obstáculo. En el caso de que exista, el robot se detendrá hasta que el obstáculo desaparezca.
- La cuarta y última tarea consistirá en comprobar los sensores sigue lineas que incorpora el robot Maqueen cada 100ms, de esta forma, cada 100ms comprobaremos que resultado nos devuelven estos dos sensores y podremos decidir en que dirección debe de ir nuestro robot para cumplir con el cometido de seguir las lineas pintadas o trazadas en el suelo.

Para programar una tarea en M2OS es necesario crear un fichero '.ads' y un fichero '.adb'. En el fichero '.ads' tendremos la especificación de nuestra tarea la cual únicamente incorpora la compartición del stack de M2OS y define el cuerpo de la tarea que vamos a crear en el fichero '.adb' de la siguiente forma:

```
with AdaX_Dispatching_Stack_Sharing;

package Maqueen_Task_Lights is
    pragma Elaborate_Body;
end Maqueen_Task_Lights;
```

Por otro lado, y como ya mencione antes, en el fichero '.adb' tendremos el cuerpo de nuestra tarea, es decir, el trabajo funcional que nuestra tarea va a ejecutar cuando se llame cada período específico de tiempo. El código que siguen las tareas de M2OS se detalla a continuación tomando como ejemplo el cuerpo de la tarea sigue lineas del Maqueen.

```
..
-- Añadir al código todas las librerías requeridas.
with Ada.Real_Time;
with M2.Direct_IO;
with M2.HAL;
with M2.Kernel.API;
with Maqueen;

-- Inicializar el cuerpo de la tarea
package body Maqueen_Task_Patrol is
   package DIO renames M2.Direct_IO;
   package RT renames Ada.Real_Time;

   use type RT.Time_Span;
```

```
-- Establecer el periodo con el que se va a ejecutar la tarea (En este caso 50ms)
   Period : constant RT.Time_Span := RT.Milliseconds (50);

   Next_Time : RT.Time;
   Loop_Count : Natural := 0;

-- Inicializar un procedimiento que se ejecutará
-- cuando se inicie la tarea por primera vez.
   procedure Other_Task_Init is
   begin
      Next_Time := Ada.Real_Time.Clock;
      Maqueen.Initialize;
      Maqueen.SetSpeed(50);
      DIO.Put_Line ("-Maqueen Initialization-");
   end Other_Task_Init;

-- Establecer que instrucciones ejecutará nuestra
-- tarea en la inicialización de la misma.
   procedure Other_Task_Body is
   -- Inicializar el proceso que contendrá el código que ejecutará nuestra tarea.
      Left_Sensor_Value : Boolean := False;
      Right_Sensor_Value : Boolean := False;
   begin
      Left_Sensor_Value := Maqueen.Read_Patrol_Left;
      Right_Sensor_Value := Maqueen.Read_Patrol_Right;

      if(Maqueen.Is_Running) then
         if(Left_Sensor_Value = False and Right_Sensor_Value = False) then
            Maqueen.Forward;
         end if;

         if(Left_Sensor_Value = True and Right_Sensor_Value = False) then
            Maqueen.Turn_Left;
         end if;

         if(Right_Sensor_Value = True and Left_Sensor_Value = False ) then
            Maqueen.Turn_Right;
         end if;

         if(Left_Sensor_Value = True and Right_Sensor_Value = True) then
            Maqueen.Spin_Left;
         end if;
      end if;

-- Suspender la tarea hasta el siguiente punto de activación (Momento actual + Periodo)
      Next_Time := Next_Time + Period;
      delay until Next_Time;
   end Other_Task_Body;

-- Por último, establecer para la tarea su prioridad, su procedimiento de inicialización
   Other_Task : AdaX_Dispatching_Stack_Sharing.One_Shot_Task
     (Init_Ac  => Other_Task_Init'Access,
      Body_Ac  => Other_Task_Body'Access,
      Priority => 4);
end Maqueen_Task_Patrol;
```

Este es el esqueleto que seguirán todas las tareas que programemos, con lo cual, cada tarea creada (en este caso cuatro) tendremos los ficheros de especificación '.ads' y los ficheros que definen el cuerpo de las tareas '.adb'.

Una vez tenemos todos los ficheros generados, deberemos de crear nuestro ejemplo concreto que incorpore estas tareas. De tal forma que podamos comprobar que con un único programa simultaneo podemos ejecutar estas cuatro tareas. Para ello, se creó un fichero '.adb' adicional con la siguiente estructura:

```
-- Establecemos las políticas de expulsión, en que orden se ejecutan las tareas etc..
pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
pragma Locking_Policy(Ceiling_Locking);
pragma Queuing_Policy(Priority_Queuing);

-- Añadir al programa las tareas
with M2.Direct_IO;
with Maqueen_Task_Left_Light;
with Maqueen_Task_Right_Light;
with Maqueen_Task_Sensor;
with Maqueen_Task_Left_Patrol;

-- El cuerpo de este ejemplo únicamente imprimirá por consola '-Maqueen Main-' pero
-- no realiza ningún trabajo funcional ya que, son las tareas programadas las
-- encargadas de ejecutar el código funcional
procedure Maqueen_Task  is
   package DIO renames M2.Direct_IO;

begin
   DIO.Put_line ("-Maqueen Main-");
end Maqueen_Task;
```

Si ejecutamos en nuestro micro:bit el programa anterior, tendremos un programa concurrente de mediana complejidad ejecutándose sobre M2OS y saltándose el límite que teníamos con el soporte oficial de AdaCore que únicamente nos permitía ejecutar 3 tareas de forma simultánea. Con lo que, el objetivo principal del proyecto queda realizado.

8. Conclusiones y Trabajos futuros

8.1. Conclusiones

Este proyecto tenía el objetivo de realizar el portado del sistema operativo M2OS diseñado por la Universidad de Cantabria al microcontrolador micro:bit. Para ello, nos hemos apoyado en la librería Ada Drivers Library y tomamos como base el portado de M2OS a la placa SMT32F4.

Los microcontroladores son unos dispositivos muy utilizados tanto a nivel industrial, como a nivel educativo ya que otorgan gran cantidad de posibilidades con un presupuesto bastante bajo. Además, cada vez se necesita en mayor medida que estos dispositivos realicen una mayor cantidad de tareas de forma simultanea, pero, como estos dispositivos no tienen memoria de sobra, es necesario tener en cuenta como vamos a gestionarla.

Este es uno de los motivos por los cuales este proyecto ha sido muy interesante debido a que he podido experimentar con el mundo de la robótica, de las tareas y además, hacer compatible un sistema operativo en este microcontrolador para futuros alumnos.

Se ha conseguido realizar una implementación totalmente funcional, la cuál pasará a ser parte de la distribución M2OS y con un gran potencial de ser una plataforma donde futuros alumnos de la Universidad de Cantabria puedan desarrollar prácticas con las que mejorar sus conocimientos en robótica y aplicaciones concurrentes.

Actualmente, el proyecto es una rama del repositorio oficial de M2OS.[16]

Para dar una mayor facilidad de uso, además, se ha programado una librería para el robot Maqueen con la que se ha conseguido un código más claro y legible.

Este proyecto me ha permitido continuar desarrollando mis conocimientos adquiridos a lo largo del grado de Ingeniería Informática, en especial de asignaturas como 'Sistemas de Tiempo Real' o 'Sistemas Operativos Avanzados', ambas de la mención de ingeniería de computadores.

Por último, el poder podido aportar al soporte oficial de AdaCore para micro:bit una parte de mi código ha sido una de las cosas más importantes del proyecto ya que, gracias a esto, una parte de mi código perdurará de forma oficial en el repositorio de Ada Drivers Library.

8.2. Trabajos futuros

Expongo en las siguiente lineas, unas propuestas de mejora para la integración en la versión oficial de M2OS.

- Implementar las funciones para gestionar el stack en el m2-hal.adb.
- Medir las prestaciones de M2OS sobre microbit en 'tests/performance/' y comparación con GNAT.
- Hacer funcionar los tests de M2OS con el emulador QEMU.

Bibliografía.

[1] Wikipedia. *Origen e Historia del microcontrolador micro:bit.* URL: https://es.wikipedia.org/wiki/Micro_Bit#Historia.

[2] AdaCore. *Ada Drivers Library.* URL: https://github.com/AdaCore/Ada_Drivers_Library.

[3] Tech.Microbit. *Datasheet Microbit V1.3.x.* URL: https://tech.microbit.org/hardware/1-3-revision/.

[4] AdaCore. *Ada, The language for safe, secure and reliable Software.* URL: https://www.adacore.com/about-ada.

[5] AdaCore. *Download ARM ELF version of GNAT Compiler.* URL: https://www.adacore.com/download.

[6] DFRobot. *DFRobot.* URL: https://www.dfrobot.com/about-us.

[7] LeRoy Miller. *micro-Maqueen-Arduino-Library.* URL: https://github.com/kd8bxp/micro-Maqueen-Arduino-Library.

[8] Mike Spivey. *Librería baremetal-v1.* URL: https://github.com/Spivoxity/baremetal-v1.

[9] HetPro-Store. *I2C.* URL: https://hetpro-store.com/TUTORIALES/i2c/.

[10] AdaCore. *GNAT Studio. The simple powerful IDE.* URL: https://www.adacore.com/gnatpro/toolsuite/gnatstudio.

[11] Cutecom. *Cutecom.* URL: http://cutecom.sourceforge.net/.

[12] Mario Aldea Rivas y Hector Perez Tijero. "Leveraging real-time and multitasking Ada capabilities to small microcontrollers". En: *Journal of Systems Architecture* 94 (2019), págs. 32-41. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.02.015. URL: https://www.sciencedirect.com/science/article/pii/S1383762118302212.

[13] Universidad de Cantabria. *M2OS. RTOS with simple tasking support for small microcontrollers.* URL: https://m2os.unican.es/.

[14] AdaCore. *Customizing through XML and Python files (15.5).* URL: https://docs.adacore.com/gps-docs/users_guide/_build/html/extending.html.

[15] Nordic Semiconductir. *nRF51 Series Reference Manual v3.0.* URL: https://infocenter.nordicsemi.com/pdf/nRF51_RM_v3.0.pdf.

[16] Mario Aldea. *Repositorio GitHub M2OS.* URL: https://gitlab.com/marioaldea/M2OS/-/tree/microbit.

[17] AdaCore. *Descarga gnat-2021-20210519-x86$_6$4 − linux − bin.* URL: https://community.download.adacore.com/v1/f3a99d283f7b3d07293b2e1d07de00e31e332325?filename=gnat-2021-20210519-x86_64-linux-bin&rand=1467.

[18] AdaCore. *Descarga gnat-community-2018-20180524-arm-elf-linux64-bin.* URL: https://community.download.adacore.com/v1/b8be4951d12b8fd0b033b8d6c5002f3a42ea9722?filename=gnat-community-2018-20180524-arm-elf-linux64-bin&rand=761.

A. Instalación del compilador y el IDE de GNAT.

Para la instalación del compilador y del GNAT Studio Programming utilizado en el proyecto se seguirán los siguientes pasos:

1. En la pagina principal de AdaCore se pueden encontrar los paquetes necesarios para la instalación:

   - GNAT Studio Programming: Para la descarga de la interfaz de programación 2021 iremos a la pestaña de descargas y, dependiendo de la arquitectura de nuestro SO (32 o 64 bits) elegiremos la versión que corresponda.[17]

   - Compilador: Para descargar la versión 'ARM ELF' del compilador, en la misma página donde descargamos previamente el IDE de GNAT, buscaremos ARM ELF (hosted on linux) del año 2018 para proceder a descargarlo.[18]

2. Una vez descargados tanto el IDE como el compilador, será necesario la instalación de algunos elementos adicionales como libncurses5, python etc...

   ```
   $ sudo apt install libncurses5
   $ sudo apt-get install python3-pip
   $ python3 -mpip install -U pyocd
   ```

3. Una vez instalados todos los requisitos previos, tocará instalar el IDE y el compilador. Para ello vamos a la carpeta de descarga y ejecutamos los dos archivos .bin realizando antes un chmod +x para dar permisos de ejecución.

   ```
   $ chmod +x gnat-community-2018-20180524-arm-elf-linux64-bin
   $ chmod +x gnat-2021-20210519-x86_64-linux-bin
   $ ./gnat-community-2018-20180524-arm-elf-linux64-bin
   $ ./gnat-2021-20210519-x86_64-linux-bin
   ```

Para poder utilizar de una forma comoda la interfaz de programación de GNAT, añadiremos ambos directorios donde hemos instalado el IDE y el compilador como variables de entorno, para ello, utilizaremos el comando export:

```
'export PATH=~/opt/GNAT/2021-arm-elf/bin:~/opt/GNAT/2021/bin:$PATH'
```

Un error conocido tras la instalación del IDE es que al intentar flashear un programa al microcontrolador micro:bit obtenemos el siguiente error: Error opening serial port ...

Este fallo es debido a una incorrecta configuración de los permisos de los puertos serie, para ellos se debe comprobar nuestro caso y corregirlo. Para ello se puede ejecutar el siguiente comando.

```
$ sudo chmod a+rw /dev/ttyACM0
```

Donde le estamos dando permisos de lectura y escritura sobre el dispositivo que, en mi caso, corresponde al microcontrolador micro:bit.

Una vez realizados estos pasos, tenemos el entorno de trabajo listo para comenzar.

B. Instalación de la librería 'Ada Drivers Library'.

Una vez ya instalado el IDE y el compilador para trabajar con M2OS y micro:bit,se procederá a la instalación de la librería 'Ada Drivers Library'. Para ello se deben seguir los siguientes pasos:

1. La librería se puede encontrar en un repositorio github publico por lo que se utiliza el siguiente comando para descargarla.[2]

   ```
   $ git clone https://github.com/AdaCore/Ada_Drivers_Library
   ```

2. Para una primera prueba, podemos colocar la carpeta en el directorio que queramos. Sin embargo, para que funcione correctamente esta librería con M2OS será necesario colocar la carpeta descargada del repositorio en la siguiente ruta de M2OS:

   ```
   $ /M2OS-microbit/arch/microbit
   ```

C. Uso de GitHub a lo largo del proyecto

A lo largo del proyecto se ha ido llevando un control de las modificaciones realizadas con GitHub. GitHub es plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador.

A lo largo del proyecto he trabajado sobre el repositorio de mi profesor Mario Aldea, haciendo un clone de la rama Microbit:

```
$ git clone https://gitlab.com/marioaldea/M2OS.git
```

Sobre ese repositorio iremos a la rama de Microbit con el comando:

```
$ git checkout microbit
```

Para subir los cambios al repositorio git se deben de ejecutar los siguientes comandos:

```
$ git add archivo/s o directorios
$ git commit -m "Comentario en ingles"
$ git push origin microbit
$ (Meter nuestro usuario y contraseña de GitHub)
```

Además, podemos ejecutar el comando 'meld .' en el directorio git para ver los ficheros modificados.

Si realizamos cualquier cambio sobre un fichero y ejecutamos un 'git status' podremos ver todos los ficheros que hemos ido modificando desde la ultima vez que hicimos una subida al repositorio. De esta forma podemos llevar un control de las modificaciones e incluso restaurar alguna versión anterior en caso de error.