

MICROPROCESADORES

PRÁCTICA 1

ENTORNO KEIL μ VISION 5, LENGUAJE DE ENSAMBLE, INTERFACES BÁSICAS DE I/O Y GESTIÓN DE EVENTOS

TITULACIONES DE GRADO DE LA
ETSI DE TELECOMUNICACIÓN



DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

PRIMAVERA 2022 - 2023

© 2022-23 DTE - UPM

ÍNDICE

1. PLANIFICACIÓN	4
2. OBJETIVOS	4
3. INTRODUCCIÓN	5
4. ESTRUCTURA DE CARPETAS	7
5. TRABAJOS PREVIOS A LA PRIMERA SESIÓN PRESENCIAL	8
5.1. Instalación de la herramienta Keil μVision 5	8
5.2. Instalación de la licencia MDK-Community	13
5.3. Instalación de <i>drivers</i> y actualización del <i>firmware</i>	13
5.4. Instalación de <i>software</i> adicional	14
5.5. Ejercicio 0 - Montaje del circuito	14
5.5.1. MATERIALES NECESARIOS PARA LA REALIZACIÓN DE LAS PRÁCTICAS	14
5.5.2. CÁLCULO Y MONTAJE DE LEDS Y DISPLAYS DE 7 SEGMENTOS	15
5.5.3. VERIFICACIÓN DEL MONTAJE Y CARGA DE SOFTWARE	18
5.6. Ejercicio 1 - ciclo de trabajo con Keil μVision 5 (ensamble)	18
5.7. Ejercicio 2 - puntos de ruptura	23
5.8. Otros ejercicios de lenguaje de ensamble	27
5.9. Ejercicio 3 - Ciclo de trabajo con keil μvision 5 (C/C++)	28
5.10. Ejercicio 4 - Eventos	37
5.11. Ejercicio 5 - Control del <i>display</i> de 7 segmentos	38
5.11.1. ADICIÓN DE GROUPS, FICHEROS Y LIBRERÍAS A UN PROYECTO	39
5.11.2. REQUISITOS PARA ESTA APLICACIÓN	41
5.12. Ejercicio 6 - Gestión de varios eventos	42
6. TRABAJOS PREVIOS A LA SEGUNDA SESIÓN PRESENCIAL	44
6.1. Ejercicio 7 - Montaje del circuito	44
6.1.1. MONTAJE DE PULSADORES Y SEGUNDO <i>DISPLAY</i>	44
6.1.2. VERIFICACIÓN DEL MONTAJE Y CARGA DE SOFTWARE	46
6.2. Ejercicio 8 - Gestión de los interruptores	46
6.3. Ejercicio 9 - Multiplexación «lenta» de <i>displays</i>	47

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

6.4.	Ejercicio 10 - Trabajo con varios ficheros fuente	47
6.5.	Ejercicio 11 - Multiplexación de <i>displays</i>	49
6.6.	EJERCICIO 12 - AAPCS	51
6.7.	Ejercicio 13 - Uso de <code>printf()</code> para depuración	55
7.	SUBIDA DE RESULTADOS A MOODLE	59

1. PLANIFICACIÓN

En este enunciado se describen los trabajos a realizar durante esta primera práctica. La práctica dispone de dos sesiones presenciales de laboratorio. Debe tener en cuenta que las sesiones presenciales de laboratorio son, más bien, sesiones de tutoría colectiva en las que el docente resolverá las dudas que durante la realización (previa a la sesión) de la práctica le hayan podido surgir. El grueso del trabajo práctico del laboratorio se espera que sea anterior a las sesiones presenciales de laboratorio y que usted organice ese trabajo de la forma que le sea más conveniente. Debe presupuestar un trabajo previo a cada sesión de laboratorio de al menos cuatro horas.

2. OBJETIVOS

Los objetivos fundamentales de esta práctica son:

- instalar en su propio PC el entorno de desarrollo *Keil μ Vision 5* para microcontroladores basados en procesadores de la familia *Cortex-M*, lo que le permitirá trabajar los contenidos de la asignatura fuera de las sesiones presenciales del laboratorio y, también, le será de ayuda durante la resolución de problemas de la misma, tanto de lenguaje de ensamble como de temas posteriores en los que se trabaja con C/C++;
- montar parte del circuito que se usará a lo largo de las restantes prácticas;
- iniciar el estudio de las técnicas de programación útiles en el desarrollo de aplicaciones para microcontroladores y sistemas empuotrados, en este caso la gestión de eventos (otras técnicas como son las *interrupciones* y las *máquinas de estados finitos controladas por eventos* se estudiarán en prácticas posteriores, pero debe saber que estas técnicas se basan en la gestión de eventos que se verá en esta práctica);
- aprender el manejo del depurador (*debugger*) y otras ayudas para la depuración durante el desarrollo de aplicaciones para sistemas empuotrados;
- aplicar el AAPCS, de modo que comprenda cómo es posible crear aplicaciones en las que parte de código se escribe en C/C++ y otra

parte en lenguaje de ensamble. Esta parte le permitirá comprender mejor el nexo que une el *hardware* del procesador con el *software* de alto nivel que corre sobre él.

3. INTRODUCCIÓN

En el ámbito de los microprocesadores, se denomina *entorno de desarrollo* al conjunto de aplicaciones que permiten la creación y la verificación del funcionamiento de los programas que ejecutará el microprocesador para gobernar un sistema electrónico construido en base a él.

Uno de los entornos de desarrollo más utilizados profesionalmente es el ofrecido por la firma *Keil* (ahora propiedad de arm). Los entornos de *Keil* integran varias herramientas informáticas que permiten crear, simular y depurar los programas escritos en lenguaje C/C++ o en lenguaje de ensamble para distintos microprocesadores, entre ellos casi todos los de la línea *Cortex-M* de arm.

Las herramientas de desarrollo de *Keil* permiten realizar y depurar programas para distintas familias de microcontroladores:

- MDK-Arm: para dispositivos basados en procesadores arm *Cortex-M*, es la herramienta usada en el laboratorio y soporta además dispositivos basados en ARM7, ARM9 y *Cortex-R*;
- Arm Development Studio: para sistemas basados en procesadores arm de prácticamente cualquier familia;
- C166: para dispositivos basados en C166, XC166, y XC2000;
- C251: para dispositivos derivados del microcontrolador 80251;
- PK51: para dispositivos derivados del 8051.

Con excepción del entorno Arm Development Studio (basado en *Eclipse*), el resto de entornos de *Keil* se basan en su propia herramienta *µVision*, actualmente en su quinta generación *µVision 5*, particularizada en cada caso con los compiladores, ensambladores y depuradores para cada familia de procesadores. Esta será la herramienta usada para la realización de las prácticas de este laboratorio. Puede encontrar más información sobre estas herramientas en la página web de *Keil*: <http://www.keil.com/>

Habitualmente se aprende a programar realizando programas que se ejecutan sobre un ordenador personal PC; es decir se realizan progra-

mas que finalmente van a ser ejecutados por el propio microprocesador (usualmente *Intel* —actualmente familias *Core i*—, aunque como sabrá, algunos fabricantes ya incorporan procesadores basados en arquitecturas *arm* a sus *PC*) que incorpora el *PC* en el que se han desarrollado. Los programas realizados se ejecutan además bajo un sistema operativo (*MS-DOS*, *Windows*, *Linux*, *mac OS* —derivado de *UNIX*—, etc.) que ya está ejecutándose en el ordenador. El sistema operativo proporciona funciones de acceso a dispositivos de *I/O* (discos, puertos de comunicaciones, teclado, monitor, etc.), de manejo de la memoria, etc.

Al programar un microcontrolador, en cambio, el programa que se realiza muchas veces es el único *software* que ejecutará el microcontrolador, es decir no hay sistema operativo y todo el control y manejo del *hardware* se hace desde el propio programa que se está realizando. En estas ocasiones se utiliza también un *PC* como herramienta para la creación y depuración de los programas, pero la ejecución final de la aplicación será en el microcontrolador (en un *hardware* diferente, por tanto) el encargado de realizarla.

Para facilitar la tarea de depuración los fabricantes de microcontroladores proporcionan *simuladores* para *PC* que permiten probar los programas sin necesidad de disponer del *hardware* en el que se encuentra el microcontrolador. Uno de estos simuladores es el que se empleará para una parte de esta primera práctica. Hay que hacer notar que muchas veces estos simuladores no simulan *exactamente* el comportamiento del microcontrolador, habiendo algunas facetas del mismo, por ejemplo los dispositivos de *I/O* que incluya, para los que la simulación puede no recoger todos los detalles del dispositivo real. En esta práctica el simulador se empleará para simular pequeños programas escritos en lenguaje de ensamble, lo que le será de utilidad durante el estudio de dicho lenguaje y para la resolución de problemas relacionados.

En una primera fase de esta práctica (en lenguaje de ensamble) la depuración del programa se va a realizar mediante el simulador del microcontrolador que incorpora el entorno de desarrollo. Posteriormente, en una segunda fase, se trabajará ya con lenguajes de programación de más alto nivel (*C/C++*) y se ejecutarán los programas en el microcontrolador real y se comprobará el funcionamiento de dichos programas mediante un *depurador*. En ambas fases se comprobará el funcionamiento del programa realizado mediante:

- La inspección de la memoria y los registros: comprobando los valores que toman las variables y los registros del microcontrolador,

modificándolos si es necesario, a medida que se ejecuta el programa.

- El control de la ejecución del programa: ejecutando cada instrucción individualmente (ejecución paso a paso), ejecutando hasta llegar a una instrucción concreta (puntos de ruptura) o bien realizando el reinicio del microprocesador (*reset*).

4. ESTRUCTURA DE CARPETAS

Los distintos archivos, librerías y proyectos de la herramienta *Keil* que se emplean en este laboratorio deben seguir, en el disco duro de los ordenadores, una cierta estructura para asegurar el correcto funcionamiento de todo el código que se entrega para las diferentes prácticas. En la figura 1 se muestra esta estructura de carpetas (en esta figura solo se incluyen las carpetas correspondientes a la práctica 1). Los detalles de esta estructura son:

- Toda la estructura aparece dentro de una carpeta MICR, aunque esta carpeta puede tener cualquier otro nombre que le sea de utilidad.
- Los proyectos de *Keil* de las tres prácticas (P1 a P3) se encuentran dentro de las carpetas P1 a P3. Estas carpetas deben estar ubicadas directamente dentro de la carpeta MICR.
- Dentro de las carpetas P1 a P3 aparecen carpetas de nombre S? (con «?» siendo una cifra o la letra «x»). Estas carpetas contienen los materiales correspondientes a las diferentes sesiones de una misma práctica (S1, S2...). Para el caso de que una cierta práctica o actividad no separe sus ejercicios en sesiones, esta carpeta se llamará Sx.
- Dentro de las carpetas S? se encuentran carpetas que contienen los proyectos de *Keil*. **Debe ser muy cuidadoso con la ubicación de las carpetas de proyectos de Keil. Es imperativo que estos proyectos se encuentren en carpetas 3 niveles por debajo de la carpeta «origen» MICR, en caso contrario las compilaciones de los proyectos no serán fructíferas debido a que el *linkador* no podrá localizar las librerías** que se describen sucintamente a continuación.

- Las carpetas *mbed* y *sw_tick_serial* contienen librerías. *mbed* se empleará en las prácticas P1 a P3, mientras que *sw_tick_serial* solo durante la práctica P1.

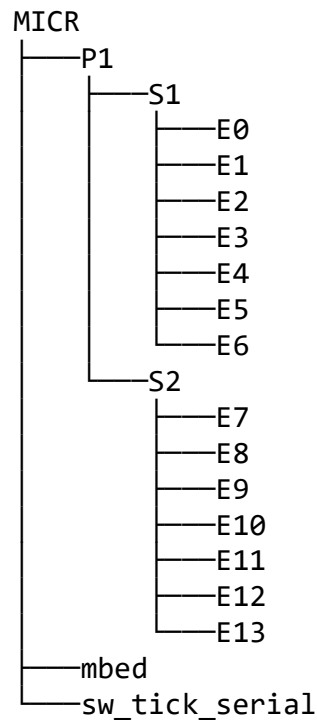


FIGURA 1: Estructura de carpetas para la primera práctica del laboratorio.

Adjunto al enunciado de esta práctica encontrará en *Moodle* un fichero comprimido que, al descomprimirlo, genera la estructura de carpetas correspondiente a la práctica 1.

5. TRABAJOS PREVIOS A LA PRIMERA SESIÓN PRESENCIAL

5.1. Instalación de la herramienta *Keil μVision 5*

La herramienta *Keil μVision 5* se encuentra instalada en los PC del laboratorio. Sin embargo, le será imprescindible instalar esta herramienta en su propio PC para poder realizar las actividades de las prácticas antes de las sesiones presenciales de laboratorio. La herramienta solo

corre en sistemas operativos del tipo *Windows*, por lo que si su PC dispone de otro sistema operativo deberá instalar alguna versión de *Windows* en una máquina virtual.

La versión del entorno que se va a emplear en el laboratorio (*Keil μ Vision 5.34*) puede ser descargada gratuitamente desde la página web de Keil (<http://www.keil.com/>, recuerde, es la herramienta MDK-Arm*). Tenga en cuenta que deberá ejecutar el instalador como *Administrador*.

En la parte final de la instalación del entorno se abre la ventana del *Pack Installer*. Esta herramienta se conecta a los servidores de Keil para descargar las últimas versiones de las herramientas de soporte para cada microcontrolador sobre el que se desee trabajar. Pasado un pequeño tiempo, al finalizar la descarga de la base de datos de microcontroladores, la ventana del *Pack Installer* tendrá el aspecto mostrado en la figura 2.

Las herramientas de soporte para cada procesador ocupan espacio en el disco duro, por lo que solamente se instalan las necesarias para los procesadores con los que se va a trabajar. En este laboratorio se hará uso de los microcontroladores:

- STM32F072RBT6: este es un microcontrolador de la firma ST *Microelectronics* basado en un núcleo arm *Cortex-M0* que se empleará para las actividades en lenguaje de ensamble (esta práctica y problemas de teoría).
- STM32L432KC: microcontrolador de la misma firma STM basado en un *core arm Cortex-M4* con unidad de punto flotante, es el microcontrolador empleado por las placas *Nucleo-l432kc* y se empleará en todas las prácticas.

Se empezará instalando el soporte para el microcontrolador STM32F072RBT6. Para ello busque, en la ventana izquierda del *Pack Installer*, el fabricante STMicroelectronics y, dentro de él (pique en «+») seleccione la serie STM32F0. Ahora, en la ventana derecha, haga *click* en el «+» de Keil::STM32F0xx_DFP y, finalmente, sobre el botón *Install* de la versión 2.1.1, tal y como se ve en la figura 3. Si en el momento en que realiza usted estas operaciones existiera una nueva versión del *pack* para este procesador, busque la versión 2.1.1 dentro del «+» de *Previous*.

* Si la versión disponible es posterior a la 5.34 también puede usarla, pero asegúrese de que las versiones de los *Packs* para los distintos procesadores son exactamente las especificadas más adelante en esta práctica.

MICROPROCESADORES

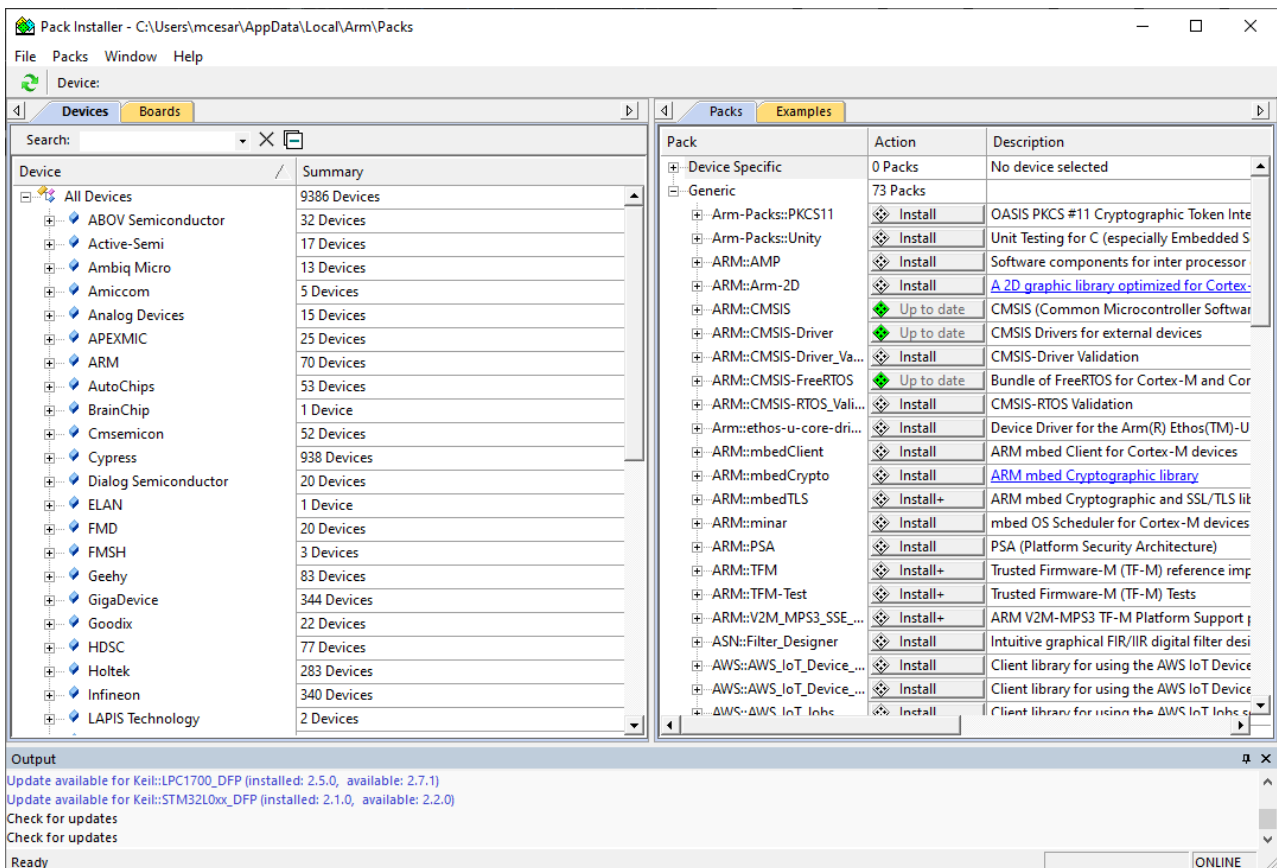



FIGURA 2: Ventana del *Pack Installer* durante la instalación de μ Vision 5.

Para el otro procesador se actúa de forma similar, aunque la versión a instalar es diferente. Concretamente, para instalar el soporte para el microcontrolador STM32L432KC, debe buscar la serie STM32L4, e instale la versión 2.6.1, como se ve en la figura 4.

Una vez instalado el soporte para estos dos microcontroladores puede cerrar el *Pack Installer*.

Para terminar la instalación de *Keil μ Vision 5* deben configurarse algunas de sus preferencias. Para ello abra *Keil μ Vision 5* ejecutándolo como administrador (botón derecho sobre el icono del programa y pique sobre  Ejecutar como administrador). Si no arrancase *Keil μ Vision 5* de esta manera los cambios que realizaría no tendrían efecto tras cerrarlo. Una vez arrancado vaya a Edit → Configuration... y ajuste en la pestaña Editor todos los campos tal como se muestra en la figura 5.

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

Asegúrese especialmente de marcar las opciones View White Space, Insert spaces for tabs (en 3 lugares) y ajuste el tamaño de las tabulaciones (Tab size) a 2 (C/C++ files), 4 (ASM files) y 2 (Other files). Una vez ajustadas todas las opciones como se ha dicho, pulse sobre OK y cierre Keil μ Vision 5. Queda ya terminada la instalación y configuración de la herramienta. Si estos últimos ajustes de la configuración del editor no se hicieran, ocurriría que la *indentación* de su código podría verse modificada si edita el código en su ordenador y luego en el ordenador del laboratorio o viceversa.

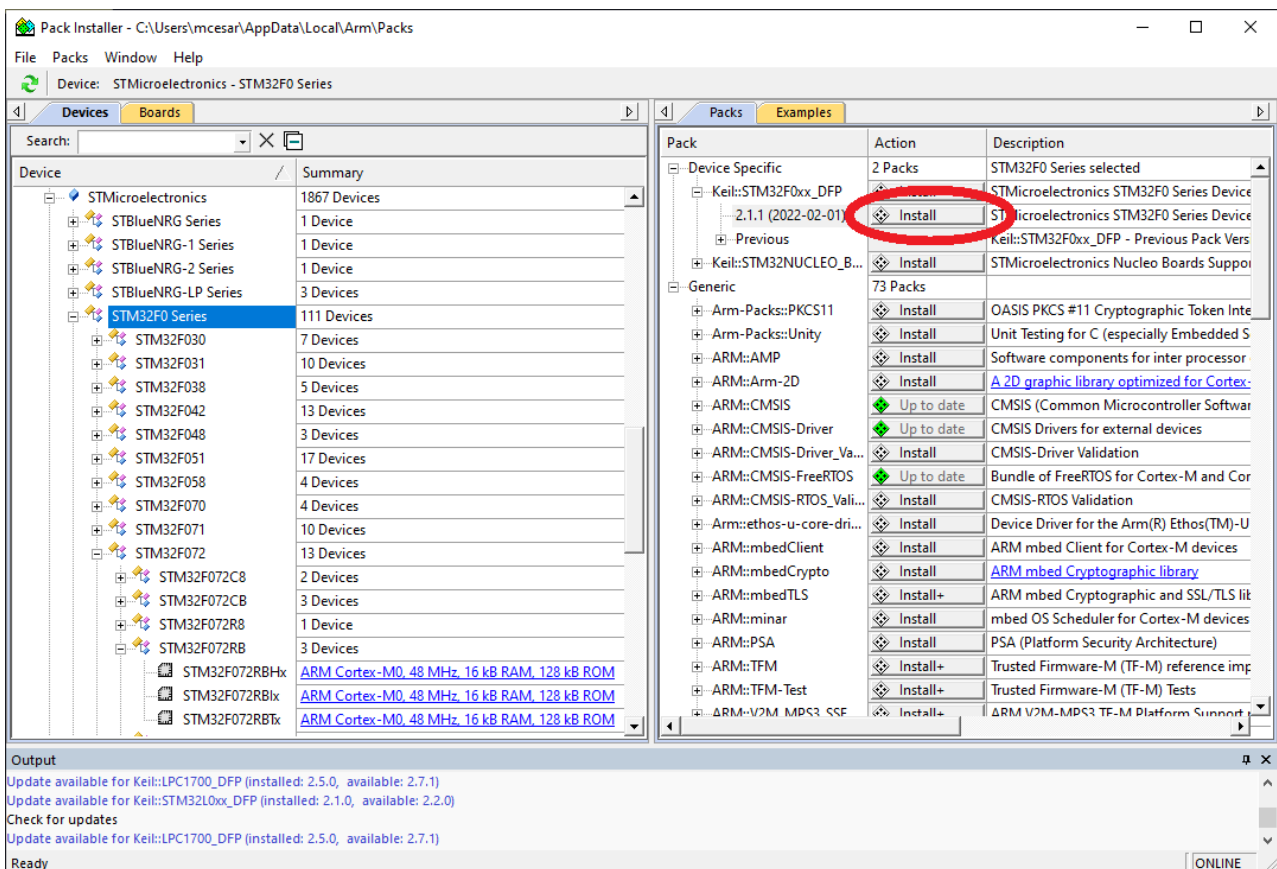


FIGURA 3: Instalación de la versión 2.1.1 del *pack* para el microcontrolador LSTM32F072RBT6.

El acceso a la placa *Nucleo-l432kc* desde μ Vision 5 (necesario para las prácticas) requiere la instalación de algún *software* adicional (*drivers* y *firmware*), cuya instalación se describe a continuación.

MICROPROCESADORES

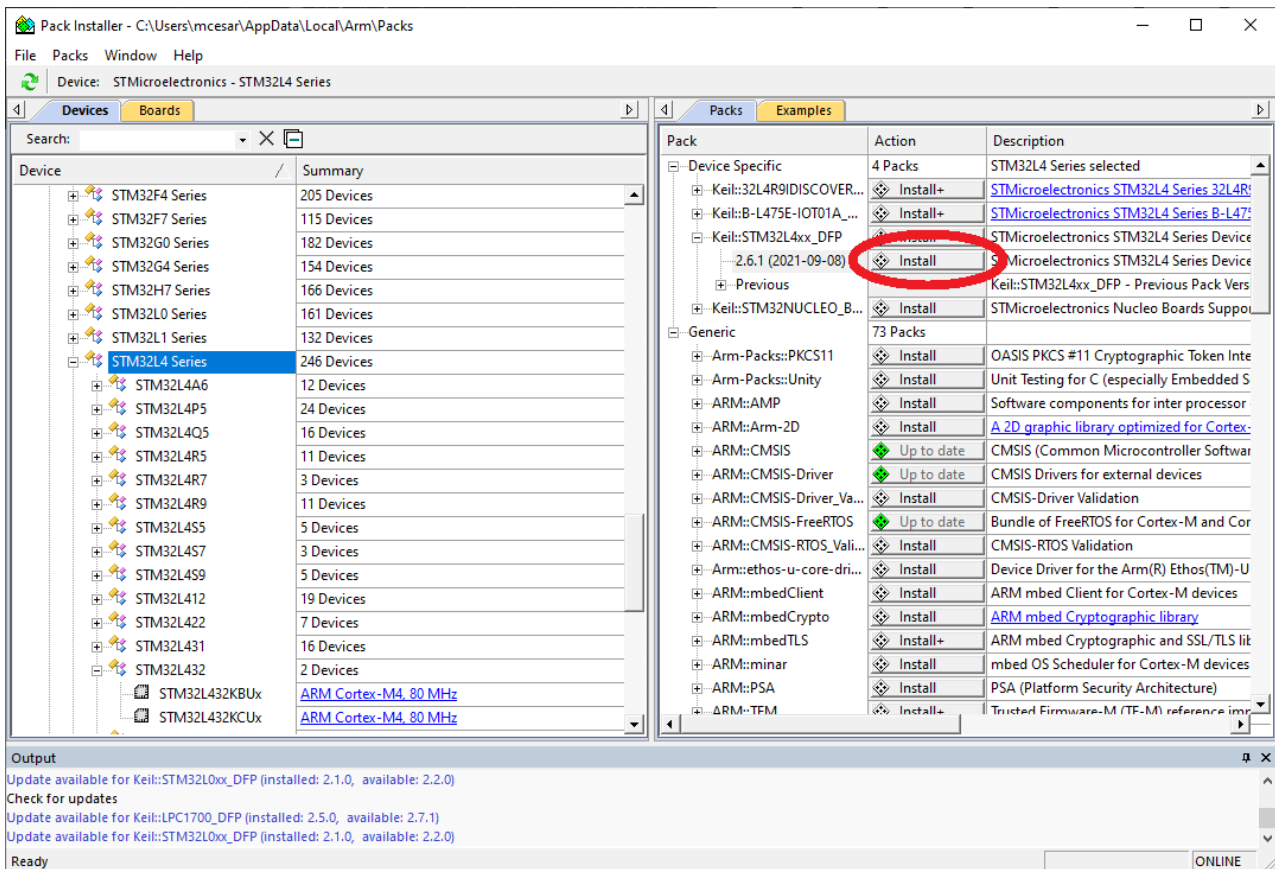


FIGURA 4: Instalación del soporte para el microcontrolador STM32L432KC.

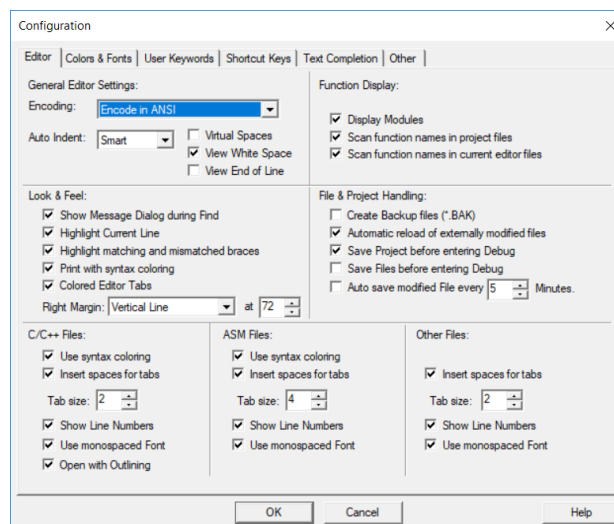


FIGURA 5: Preferencias del editor en Keil μ Vision 5.

5.2. Instalación de la licencia MDK-Community

La herramienta *Keil μ Vision 5* requiere una licencia que le permita generar ejecutables de cualquier tamaño. Sin ella el tamaño máximo de los ejecutables que puede generar es de 32 KiB, insuficiente para alguna de las prácticas de esta asignatura. La licencia necesaria se denomina MDK-Community y se puede obtener *online* a través de la URL: <https://keil.arm.com/mdk-community>. Es posible que deba crear una cuenta personal en arm para poder acceder a esta página. En ella encontrará las instrucciones necesarias para solicitar la licencia, que recibirá por email, y para incluirla en la herramienta.

5.3. Instalación de *drivers* y actualización del *firmware*

El entorno de desarrollo ya se ha instalado en el apartado anterior, sin embargo es necesario instalar un *driver* en el ordenador que permite al entorno de *Keil* comunicarse con la tarjeta. Dicho *driver* se encuentra en la carpeta Drivers (del fichero .7z adjunto a la práctica). El *driver* se instala (conecte su placa antes de ejecutar estos instalables) ejecutando (como administrador) el *script* stlink_winusb_install.bat (si este *script* fallase, emplee entonces los ejecutables dpinst_x86.exe o dpinst_amd64.exe, dependiendo de la arquitectura de su PC)

Adicionalmente, es necesario que la tarjeta tenga instalada la última versión del *Firmware** para poder usar la librería *mbed*, sobre la que se apoyan esta y las siguientes prácticas. Se encuentra en la carpeta Firmware. Una vez conectada la placa al ordenador y visible como una unidad de almacenamiento, se ejecutará el programa ST-LinkUpgrade.exe, se pulsará el botón Device Connect y, una vez reconozca la placa, se pulsará Yes para iniciar la secuencia de actualización.

* *Firmware*: *software* incluido en el producto por su fabricante y responsable de su funcionamiento. En este caso se refiere al *software* que permite a la tarjeta ser vista como un *pendrive* por el PC y también depurar los programas sobre la tarjeta a través del puerto USB.

5.4. Instalación de *software* adicional

Además de lo anterior también necesitará instalar en su PC el *software* gratuito: Tera Term (<https://ttssh2.osdn.jp/index.html.en>) y 7-Zip (<https://www.7-zip.org/>). Simplemente, descárguelos y siga sus instrucciones de instalación.

5.5. Ejercicio 0 - Montaje del circuito

El montaje del circuito que se empleará durante las prácticas se divide en varias etapas, abordándose dos de ellas en esta práctica, una previa a cada sesión.

5.5.1. MATERIALES NECESARIOS PARA LA REALIZACIÓN DE LAS PRÁCTICAS

Se relacionan a continuación los dispositivos (aparte de la placa STM) y su cable USB de conexión) que serán necesarios a lo largo de todas las prácticas de la asignatura. Todos ellos deben ser adquiridos por cada estudiante del laboratorio:

- *protoboards* para realizar el montaje (y cablecillo para realizar las conexiones y las herramientas necesarias);
- 2 *displays* de 7 segmentos de cátodo común;
- 3 LED;
- 3 pulsadores;
- 2 transistores NPN de los tipos BC547 ó 2N3904;
- 1 resistor dependiente de la luz (también denominado fotorresistor, o LDR, de *Light-Dependent Resistor*);
- algunos resistores fijos para la polarización de LED, LDR, transistores, etc.;
- 1 sensor telemétrico ultrasónico del tipo HC-SR04.

Todos estos elementos se pueden adquirir muy fácilmente a través de *Internet* o en tiendas especializadas. En esta primera práctica no se emplearán la LDR ni el sensor telemétrico.

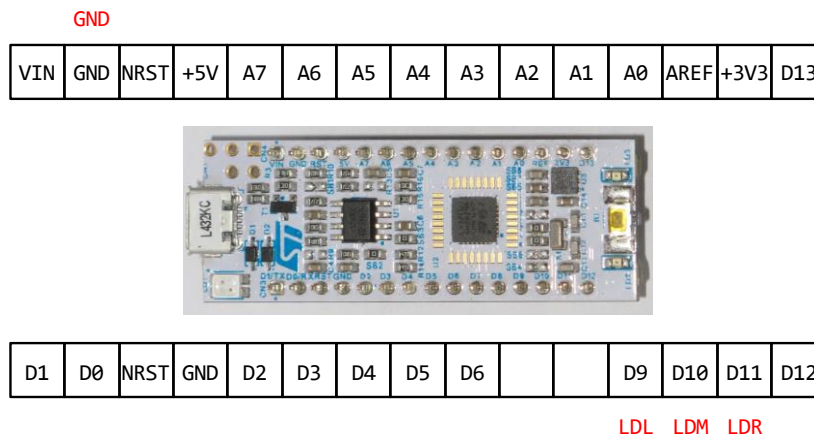
Antes de la primera sesión deberá montar los tres LED y uno de los *displays* de 7 segmentos (solo uno). Se describe ahora el montaje necesario para la primera sesión de la práctica.

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

5.5.2. CÁLCULO Y MONTAJE DE LEDS Y DISPLAYS DE 7 SEGMENTOS

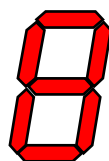
El circuito necesario para la primera sesión de esta práctica hace uso, aparte del microcontrolador, de 3 LED discretos y de un *display* de 7 segmentos (solo uno).

Cada uno de los LED estará controlado por una de las señales LDL, LDM y LDR (de *LeD Left*, *LeD Middle* y *LeD Right*). Los LED deben encenderse cuando haya un nivel lógico *alto* en la señal correspondiente. A su vez, las señales LDL, LDM y LDR se conectarán a los siguientes pines del microcontrolador STM (se incluye además masa, GND):

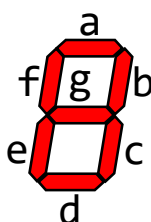


Como ya conoce (por las asignaturas Electrónica I y Electrónica II) debe incluirse una resistencia de limitación de corriente para cada uno de los LED. Los detalles de dicho montaje le fueron expuestos en la actividad individual no presencial número 1 del bloque temático 2 de Electrónica II. Los parámetros necesarios para este cálculo se encontrarán en la *datasheet* del microcontrolador (disponible en *Moodle*) y del LED concreto que haya empleado (cuya *datasheet* debe localizar usted).

Por lo que respecta al *display* de 7 segmentos (que ya empleó en la actividad de trabajo en grupo no presencial número 1 del bloque 3 de Electrónica II), éste no es más que una disposición de 7 LED, cada uno de ellos en la forma aproximada de rectángulo alargado —segmento—, que permite, según los LED que se enciendan, representar visualmente cifras decimales y otros símbolos. La disposición de los LED en estos *displays* es:



Cada LED individual (cada segmento) se designa con una letra desde la *a* hasta la *g*, según:



En la figura 6 se puede ver la representación de varios símbolos alfanuméricos en un *display* de este tipo.

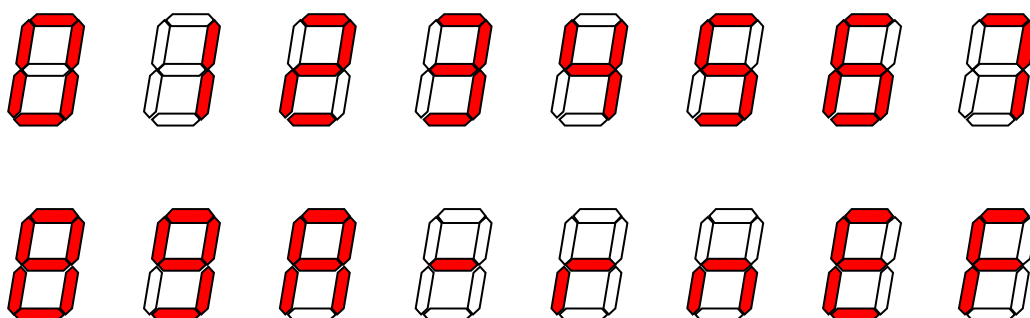


FIGURA 6: representación en un *display* de 7 segmentos de los símbolos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, -, r, n, E, F}.

Cada uno de los LED de un *display* de 7 segmentos tiene dos terminales (ánodo y cátodo), lo que daría un total de 14 pines para el *display*. Sin embargo suelen conectarse todos los ánodos o cátodos de los segmentos del *display* juntos, de modo que se dispone de *displays* de 7 segmentos:

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

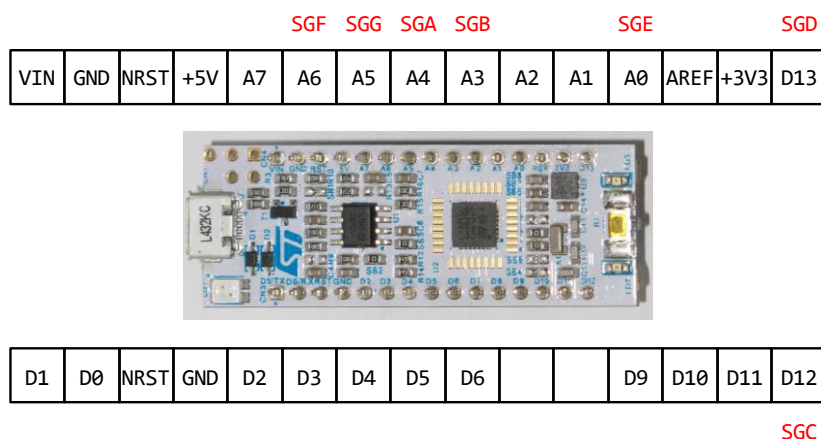
- De cátodo común, cuando todos los cátodos de los 7 segmentos se unen entre sí. Este terminal («cátodo común») se conecta a una tensión baja y los ánodos individuales de cada segmento a una tensión alta para conseguir el encendido. Este es el tipo de *displays* que debe usarse en este laboratorio.
- De ánodo común, cuando todos los ánodos de los 7 segmentos se unen entre sí. Este terminal («ánodo común») se conecta a una tensión alta y los cátodos individuales de cada segmento a una tensión baja para conseguir el encendido.

Por supuesto, cada segmento debe contar con su propia resistencia de limitación de corriente. El cálculo de las mismas deberá realizarlo usted a partir de los parámetros de la *datasheet* del microcontrolador (en *Moodle*) y del *display* de 7 segmentos que emplee (debe usted localizar su correspondiente *datasheet*).

Habitualmente los *displays* comerciales están encapsulados con 10 pines:

- siete de ellos se corresponden con los segmentos a hasta g ;
- uno más controla un punto decimal abajo y a la derecha (suele llamarse DP , de *Decimal Point*);
- y los dos últimos pines (usualmente uno arriba y otro abajo del encapsulado) son el cátodo (o ánodo, dependiendo del tipo de *display*) común.

En este laboratorio los segmentos de los *displays* se conectarán al microcontrolador mediante unas señales llamadas desde SGA hasta SGG para los SeGmentos desde el *a* hasta el *g*. Estas señales se conectarán a los siguientes pines del microcontrolador STM:



5.5.3. VERIFICACIÓN DEL MONTAJE Y CARGA DE SOFTWARE

En este apartado se cargará una aplicación de ejemplo, ya compilada, para simplemente verificar que el conexionado de los recursos es correcto. Para ello, una vez montados LED y *display* de 7 segmentos (todos ellos con las correspondientes resistencias) junto con el microcontrolador en las placas de prototipado, se conectará mediante un cable USB al ordenador. Si el *software* se ha instalado correctamente (*drivers* y *firmware*), deberá aparecer una nueva unidad de almacenamiento correspondiente a su microcontrolador. Copie el fichero ejecutable .bin adjunto a la práctica (carpeta MICR\P1\S1\E0) en la unidad. Si el cableado es correcto deberán aparecer secuencialmente, en el *display* de 7 segmentos, a una cadencia de uno por segundo, los símbolos de la figura 6* de la página 16. A la vez, los LED mostrarán, a la misma cadencia, la cuenta binaria ascendente de 3 bits.

5.6. Ejercicio 1 - ciclo de trabajo con Keil μ Vision 5 (ensamble)

En este ejercicio se presenta el manejo mínimo de la herramienta Keil μ Vision 5 y el uso de las herramientas básicas de depuración que permiten la ejecución paso a paso de un programa, así como visualizar—y modificar— el contenido de los registros y la memoria del microcontrolador. Se empleará lenguaje de ensamble y el simulador, aunque más adelante se volverá a abordar este mismo manejo de la herramienta, pero empleando lenguajes de programación de más alto nivel (C/C++) y el depurador.

Antes de comenzar a explicar la funcionalidad a implementar en esta práctica, es necesario conocer brevemente el proceso de desarrollo de

* Este programa emplea, para representar el símbolo «1», una configuración de segmentos diferente a la dada en la figura 6 de la página 7 (enciende los dos segmentos de la izquierda). Todos los ejecutables que le proporcionaremos emplean esta representación alternativa del «1» y lo hacemos para que, de un vistazo, los docentes podamos saber si el programa que se está corriendo es el entregado con la práctica o el realizado por usted. Sus programas deben representar el «1» como aparece en la figura 6.

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

aplicaciones empleando el entorno de desarrollo de *Keil uVision 5*. Haga doble *click* sobre el fichero MICR\P1\S1\E1\MICR.uvprojx (que se encuentra en el descargable —en Moodle— para esta práctica), se abrirá un entorno de desarrollo como el mostrado en la figura 7.

Este proyecto ya se encuentra configurado para trabajar con el microcontrolador STM32F072RBT6. Este es un microcontrolador de la firma STM basado en un procesador arm *Cortex-M0*, su frecuencia máxima de reloj es de 48 MHz y dispone de 128 KiB de memoria *Flash* (a partir de la dirección 0x0800 0000, que puede también direccionarse en 0x0000 0000) y 16 KiB de memoria SRAM (dirección 0x2000 0000), todo ello (junto con varios periféricos genéricos: *timers*, convertidores, GPIO, etc.) en un encapsulado de 64 pines. Se emplea para esta práctica ya que el lenguaje de ensamble de los procesadores basados en *Cortex-M0* es más simple que el de la familia *Cortex-M4* de la placa STM.

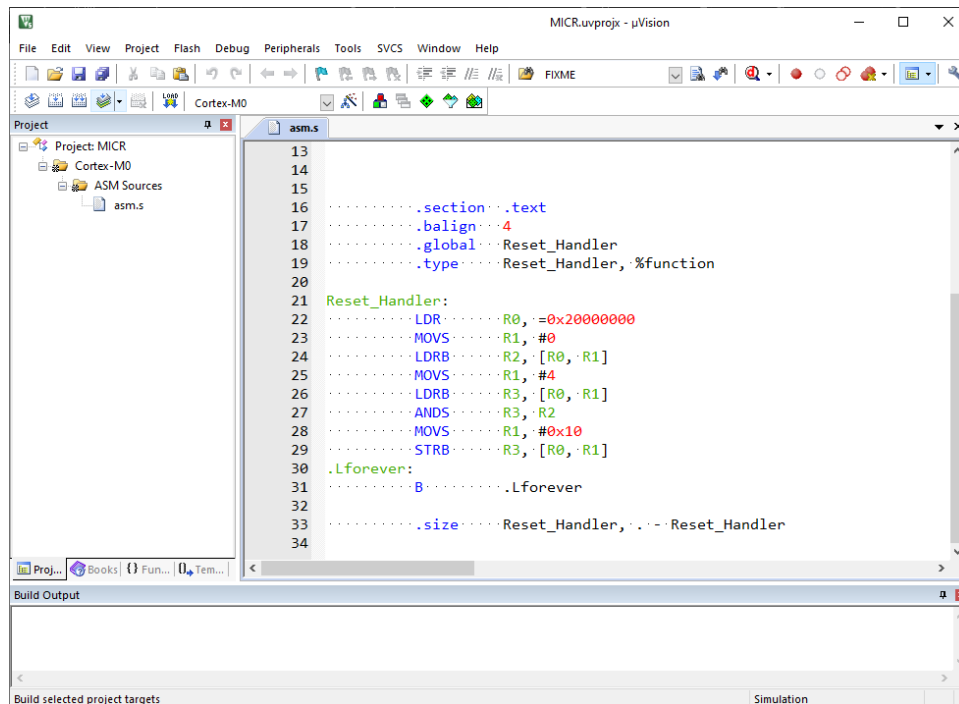




FIGURA 7: Entorno de desarrollo *Keil uVision 5*.



Este proyecto ya se encuentra configurado para poder trabajar con el simulador. Posteriormente se empleará la tarjeta para ejecutar los programas, en cuyo caso la depuración se hará con el *debugger*, en vez del simulador. En caso de querer empezar el desarrollo de un proyecto desde cero sería necesario realizar varias configuraciones del entorno de desarrollo.

En la parte izquierda se encuentran los ficheros que componen el proyecto, en este caso sólo aparece el fichero `asm.s`. Analice el código de la aplicación y comprobará que se trata de un sencillo programa que realiza la función AND de los datos almacenados en dos posiciones de memoria y guarda el resultado en otra posición. Una vez realizada esta funcionalidad el programa se queda en un bucle infinito.





Adicionalmente, el programa comienza con una serie de directivas del ensamblador que indican las posiciones de memoria en la que se almacena la pila y el código (`Reset_Handler`). Ambas son las primeras direcciones de la tabla de vectores (`.Lectors`). Más detalles sobre el funcionamiento de esta tabla se explicarán en las clases de teoría. También se define un área de código (`.text`) en la que se incluye la etiqueta `Reset_Handler`, que es donde se iniciará la ejecución.

Una vez analizado (o editado) el programa es necesario ensamblarlo para detectar posibles errores. Para ello debe emplear estos tres iconos que se encuentran en la parte superior izquierda . El primero ensambla (compila, para el caso de que el código fuente sea C/C++) sólo el fichero activo, el segundo ensambla (o compila) y enlaza (*linka*) los ficheros que hayan cambiado desde el último ensamblado (compilación) y el tercero ensambla (compila) y *linka* todo de nuevo. Normalmente sólo es necesario emplear el segundo. Como resultado del proceso, y si no hay errores en el programa, se genera un fichero ejecutable que puede correr en el microcontrolador. Para comprobar si todo el proceso se ha realizado correctamente se debe consultar la ventana inferior (Build Output), en la que aparecerán los mensajes de salida de ensamblador, preprocesador, compilador y *linkador*, incluyendo errores y *warnings*.

Para poder comprobar el funcionamiento de este programa se debe pasar la herramienta al modo de depuración (ver figura 8) pulsando el icono . Esto hará que el código se pueda simular (sin necesidad de tener la placa, como ahora) o transfiera a la placa para comenzar la ejecución (como se verá más adelante, depende de la configuración). El proyecto está configurado para *simular* y para que la ejecución se detenga justo en el gestor del *reset* (etiqueta `Reset_Handler`). A partir de este momento se puede:

- poner *breakpoints* (puntos de ruptura, se estudiarán en el siguiente apartado) mediante el botón  (o picando a la izquierda del número de línea);
- ejecutar el programa directamente empleando el botón .

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

- ejecutar paso a paso utilizando los botones    —el primero de estos tres hace que, si lo que se trata de ejecutar es una función, se pase a ejecutar paso a paso las instrucciones dentro de la función; el segundo hace que, si se trata de ejecutar una función, esta se ejecute como si fuera una única instrucción; finalmente el tercero de los botones hace que se ejecuten todas las instrucciones hasta salir de la función en la que se encuentra el programa— (las diferencias entre ellos son más útiles en C/C++, aquí use );

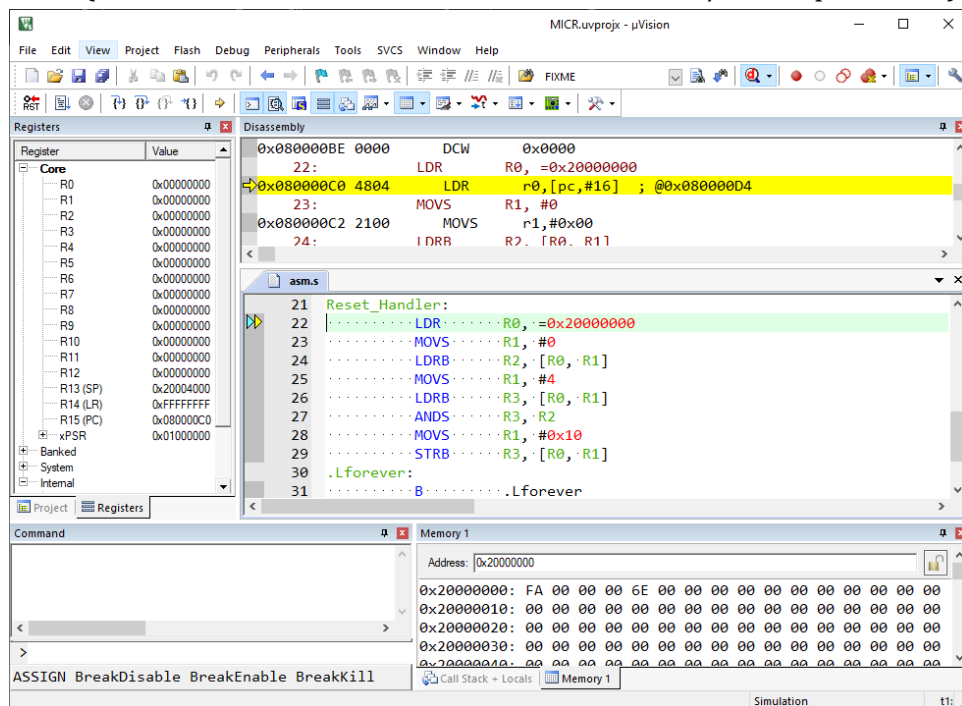








FIGURA 8: Keil μ Vision 5 en modo de depuración.




- detener la ejecución en curso mediante el botón ;
- resetear el procesador mediante el botón .
- Según se ejecuta cada paso, la flecha que aparece a la izquierda del código ( o ) se irá desplazando línea a línea. Tenga en cuenta que se puede ejecutar paso a paso en la ventana inferior (se emplea , que puede contener C/C++ o lenguaje de ensamble, en esta práctica es lenguaje de ensamble) o en la ventana superior (se emplea , siempre lenguaje de ensamble, aun cuando el código fuente sea C/C++). Para identificar qué código se está ejecutando paso a paso debe identificar cuál de las dos ventanas tiene el foco.

Antes de comenzar a ejecutar el programa es necesario tener en cuenta la información que se encuentra almacenada en las direcciones (de byte) de memoria implicadas (0x2000 0000 y 0x2000 0004). Por defecto, y como se muestra en la ventana que aparece en la parte inferior derecha de la figura (Memory 1), las posiciones implicadas están a 0. Por tanto antes de ejecutar la aplicación hay que modificar estos valores. Si no aparece la ventana de memoria porque se haya cerrado previamente, basta con ir al menú de View → Memory Window → Memory1. La dirección de inicio del visualizador de memoria se puede fijar escribiéndola en el campo Address. Es posible visualizar la memoria como bytes (por defecto) y también como datos de otros tamaños, codificados con o sin signo, en decimal, hexadecimal o ASCII. Use el botón derecho sobre el visualizador de memoria para cambiar esto.

Para poder realizar una prueba se propone que en la posición 0x2000 0000 se almacene el dato 0xFA y en la 0x2000 0004 el dato 0x6E. Para modificar el valor de una posición de memoria basta con realizar un doble *click* sobre la posición de memoria y escribirlo.

Cuando el programa está parado es posible consultar y modificar el contenido de la memoria y también los registros del procesador. El valor de los registros se muestra en la parte izquierda de la pantalla. Inicialmente se aprecia que todos los registros de propósito general (de R0 a R12) se encuentran inicializados a 0.

Una vez que todo esté configurado, es momento de comenzar a ejecutar el programa paso a paso, analizado la evolución del contenido de los registros y la memoria. Hay que recordar que el objetivo de este ejercicio es conocer cómo se depura una aplicación paso a paso, y no tanto analizar la funcionalidad del programa.

Seguidamente conviene conocer la funcionalidad de otros dos botones que aparecen en la vista de depuración. El primero permite reiniciar la ejecución del programa mediante el icono  **RST**, verá que la aplicación vuelve al comienzo y que los valores de los registros se inicializan. El segundo permite parar la ejecución de un programa que se está ejecutando de forma continua debido a que se haya pulsado sobre el botón , se trata del botón . Para comprobar su funcionalidad realice un *reset* con el botón anteriormente explicado, ejecute el programa de forma continua y pulse el botón de parar, comprobará que el programa se encuentra en el bucle infinito que hay al final del código.

Para finalizar con este ejercicio modifique el programa para que en lugar de emplear datos de 8 bits se utilicen datos de 32 bits y que en lugar de hacer la operación AND se realice la suma aritmética. Emplee como datos en memoria los números 0x9AC3 34FE (en la dirección 0x2000 0000) y 0x33A5 432D (en la dirección 0x2000 0004). Verifique mediante simulación el funcionamiento de su programa y tenga en cuenta para ello que el procesador *Cortex-M0* de esta práctica trabaja en modo *little-endian*.

5.7. Ejercicio 2 - puntos de ruptura

Este ejercicio tiene como objetivo la edición de un programa en lenguaje de ensamble y su depuración —empleando *breakpoints*— para comprobar su funcionamiento, así como observar el contenido final de algunas posiciones de memoria y valores almacenados en los registros.

Para poder realizar este apartado haga una copia del proyecto de *Keil μVision 5* del apartado anterior (es decir, todo lo que se encuentre en MICR\P1\S1\E1, excepto las carpetas cuyo nombre comience por «~») en la carpeta MICR\P1\S1\E2 y trabaje sobre esta copia. Sustituya el código, a partir de la etiqueta **Reset_Handler**, y hasta la directiva **.size** por:

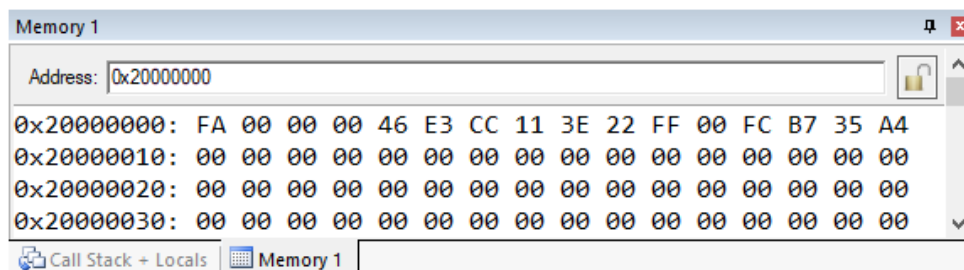
```
Reset_Handler:
    LDR        R0, =0x20000008
    MOVS       R1, #8
    EORS       R2, R2

.Lloop:
    SUBS       R1, #1
    BMI        .Lexit
    LDRB       R3, [R0, R1]
    LSRS       R3, #1
    BCC        .Lloop
    ADDS       R2, #1
    B          .Lloop

.Lexit:
    MOVS       R1, #0x08
    STRB       R2, [R0, R1]

.Lforever:
    B          .Lforever
```

Edite el programa, ensámblelo, corrija los errores que puedan aparecer y pase al modo de depuración. Adicionalmente, modifique el contenido de las posiciones de memoria que emplea el programa para que contengan los siguientes valores:



Hay que recordar que, por defecto, se emplea el simulador para ejecutar el programa. Ejecute paso a paso el programa tratando de interpretar la funcionalidad. Rellene las siguientes tablas, indicando cómo quedará la memoria y los registros del microprocesador tras la ejecución de todo el programa.

Dirección	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x2000 0000	FA	00	00	00	46	E3	CC	11	3E	22	FF	00	FC	B7	35	A4
0x2000 0010	03	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

R0	0x2000 0008
R1	0x0000 0008
R2	0x0000 0003
R3	0x0000 001F

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

Como habrá comprobado, el bucle se ejecuta 8 veces, lo que lleva a tener que realizar muchos pasos antes de llegar a la última instrucción del programa. Esto podría ser aún peor si, en lugar de 8 iteraciones, se realizaran cientos, miles o millones. Para poder abordar la depuración de estos programas es necesario poner *puntos de ruptura*. Un punto de ruptura (en inglés *breakpoint*) hace que la ejecución se pare cuando el código llega a una determinada instrucción, para poder evaluar en ella el valor de los registros o las posiciones de memoria.

Para poner un punto de ruptura se debe hacer *click* en la parte izquierda de cada instrucción, en la zona gris a la izquierda del número de línea (ver figura 9). Este *click* hará que se marque la instrucción con un círculo rojo.

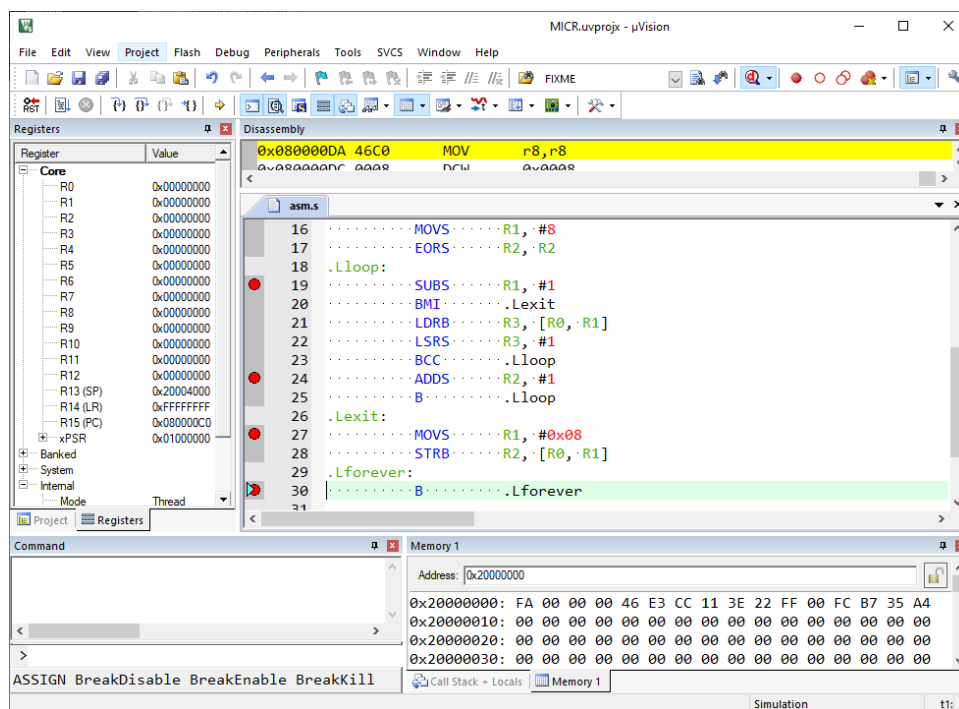





FIGURA 9: Puntos de ruptura —*breakpoints*—.

A continuación reinicie el programa con  (cuidado, *no* vuelva al modo de edición con , ya que al hacerlo se pierden todos los valores que haya introducido manualmente en la ventana de memoria) y coloque cuatro puntos de ruptura: el primero en la instrucción asociada a la etiqueta **.Lloop**; el segundo inmediatamente después de la instrucción de salto condicional a **.Lloop**; el tercero en la instrucción asociada a la etiqueta **.Lexit**; y el cuarto en la instrucción asociada a la etiqueta

.Lforever. En la figura 9 se aprecian las instrucciones en las que se deben colocar los puntos de ruptura.

Una vez establecidos los puntos de ruptura (conviene que, si ha ejecutado el programa previamente, ponga a cero el contenido de la dirección 0x2000 0010) ejecute el programa empleando el botón . Verá que la ejecución se detiene en alguno de los puntos de ruptura. A partir de ese punto puede ejecutarse paso a paso de nuevo o lanzar de nuevo la ejecución continua hasta que se llegue al siguiente punto de ruptura.

Aprovechando los puntos de ruptura que se han definido trate de averiguar si la funcionalidad que infirió anteriormente es correcta. Para ello es posible que deba modificar el contenido de algunas posiciones de memoria implicadas en el algoritmo.

En relación con los puntos de ruptura, conviene indicar que hay otras funcionalidades asociadas a ellos. Para analizarlo abra la ventana Debug → Breakpoints..., que mostrará lo visualizado en la Figura 10.

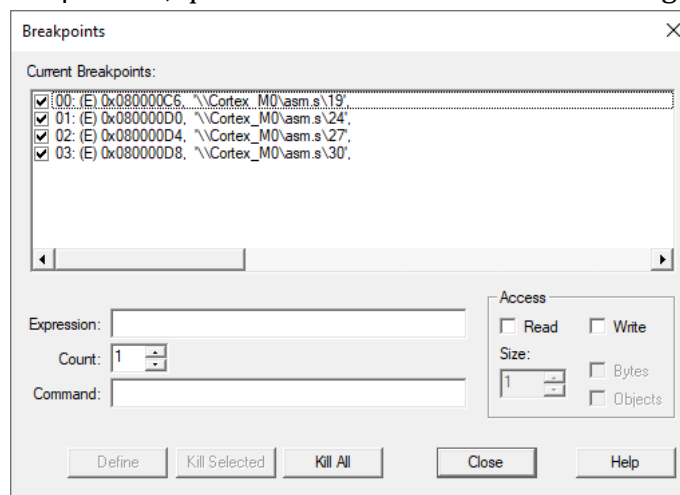


FIGURA 10: Gestión de *breakpoints*.

En esa ventana se pueden definir puntos de ruptura *condicionales* (*conditional breakpoints*), que solo detengan la ejecución en una determinada línea si, al alcanzar esa línea, se verifica además alguna condición, como por ejemplo que un registro tenga un determinado valor o que la línea haya sido alcanzada, previamente, un determinado número de veces. Es posible incluso definir puntos de ruptura sobre direcciones de memoria, de modo que la ejecución del programa se detenga cuando se accede a una determinada dirección de memoria. Por ejemplo, en la figura 11 aparecen definidos tres puntos de ruptura:

00. Punto de ruptura de *ejecución* (E) en la línea 19 del fichero asm.s.
Se corresponde con la instrucción de la etiqueta **.Lloop**. Dicha instrucción está almacenada en la dirección **0x0800 00C6** del mapa de memoria del procesador. El programa se detendrá justo antes de la ejecución de esa instrucción. Este es un *breakpoint* «normal», como los anteriores.
01. Punto de ruptura de *ejecución* (E) en la línea 24 del fichero asm.s.
Se corresponde con la instrucción **ADDS R2, #1**. Este *breakpoint* detendrá el programa justo antes de ejecutar esa instrucción por segunda (o sucesiva) vez (count=2). Dicha instrucción está ubicada en la dirección **0x0800 00D0**.
03. *Breakpoint* de *acceso* (A) que detendrá el programa cuando se *acceda* en escritura (write), por tercera o sucesiva vez (count=3) a la dirección **0x2000 0010**.

Para más información sobre el uso de *breakpoints* en μ Vision 5 consulte la ayuda del programa.

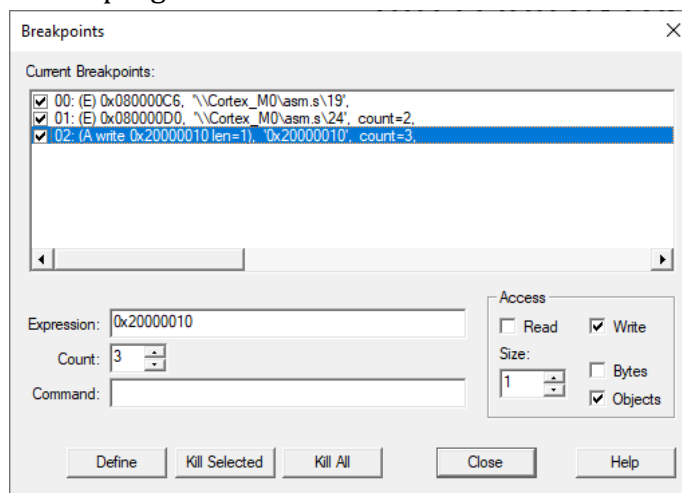


FIGURA 11: Puntos de ruptura avanzados.

Deberá averiguar la utilidad del código de este ejercicio indicando, en particular, la relación existente entre los datos inicialmente en memoria y el dato guardado en memoria por el programa.

5.8. Otros ejercicios de lenguaje de ensamble

Teniendo en cuenta todo lo anterior, se pide que emplee la herramienta *Keil μ Vision 5*, con especial énfasis en sus recursos de depura-

ción, como ayuda durante la resolución de los ejercicios de lenguaje de ensamble propuestos en las clases de teoría.

5.9. Ejercicio 3 - Ciclo de trabajo con keil μ vision 5 (C/C++)

En este apartado se pretende emplear una aplicación en C/C++, previamente desarrollada, para conocer las particularidades de la metodología de trabajo con el entorno de desarrollo de *Keil* para el caso de que se trabaje con lenguajes de alto nivel y depurando sobre una placa real, en vez de trabajar con el simulador, como hasta ahora. Para ello se debe abrir el proyecto de ejemplo que se encuentra en *Moodle* (MICR\P1\S1\E3) haciendo doble *click* sobre el *fichero de proyecto*, cuya extensión es .uvprojx. Se abrirá un entorno de desarrollo como el mostrado en la figura 12. Este proyecto ya se encuentra configurado para poder trabajar con la tarjeta del laboratorio. En caso de querer empezar el desarrollo de un proyecto desde cero, sería necesario realizar varias configuraciones del entorno de desarrollo, así como incluir la librería *mbed* en el proyecto.

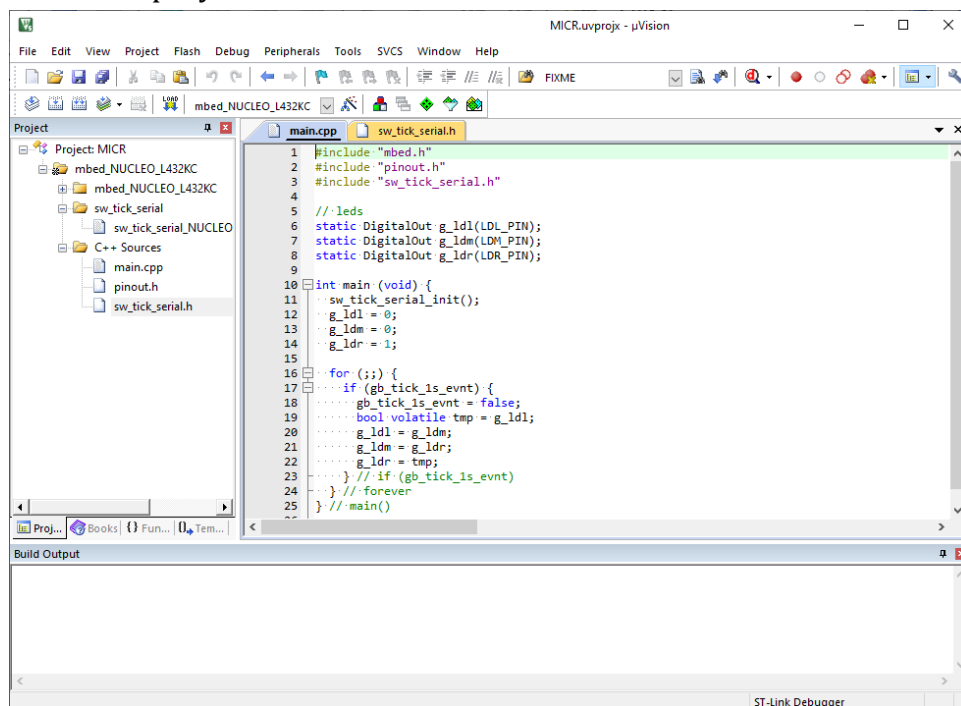



FIGURA 12: entorno de desarrollo *Keil* μ Vision 5.

Como recordará de lo dicho anteriormente y de las clases de teoría, en el caso de sistemas empuotrados es muy habitual que el desarrollo de la aplicación se haga en un ordenador (*host*) distinto al que va a, finalmente, ejecutar la aplicación (*target*). En el caso que nos ocupa, el *host* es el PC corriendo la aplicación *Keil μ Vision 5* y el *target* es la placa *STM Nucleo-l432kc*.

De nuevo en la figura 12, en la parte izquierda (dentro de la ventana Project) se encuentran los ficheros que componen el proyecto, que además se encuentran agrupados en una especie de carpetas llamadas *groups* (a diferencia de las carpetas en el disco, un *group* no puede contener otros *groups*). En este proyecto aparecen los siguientes *groups*:


- *mbed_NUCLEO_L432KC*. La librería *mbed* pre-compilada para las placas *STM Nucleo-l432kc*.
- *sw_tick_serial*. Librería *sw_tick_serial*, de uso exclusivo durante esta práctica.
- *C++ Sources*. Fuentes en C++ de la aplicación.

La aplicación final está formada por los ficheros *main.cpp* y *pinout.h*. Una vez abierto el proyecto puede editarse el código que aparece en la parte derecha. Abra el fichero *main.cpp*, que es una sencilla aplicación que enciende uno de los tres LED que se encuentran en la placa del laboratorio y va «desplazando» el LED encendido, hacia la izquierda, a una frecuencia de 1 Hz. Analice el programa, hasta la línea 8, y trate de entender su significado y cómo se relaciona este código con lo dicho en el apartado 5.5.2 en las páginas 15 y siguientes respecto a los pines de la placa STM a los que deben conectarse los LED (tendrá que consultar también el fichero *pinout.h*).

Una vez analizado (o editado) el programa es necesario compilarlo para detectar posibles errores y para generar el fichero ejecutable. Para ello debe emplear estos tres botones que se encuentran en la parte superior izquierda  y que ya fueron explicados anteriormente. Como resultado del proceso, y si no hay errores en el programa, se genera un fichero ejecutable que puede correr en el microcontrolador. Este fichero se encuentra en el PC, dentro de la carpeta *~build* en el directorio del proyecto (tiene extensión *.bin*). En los ejercicios anteriores el ejecutable fue simulado por parte del simulador que integra el entorno de *Keil*, en este caso se va a realizar la ejecución sobre el procesador real que hay en la placa, para lo que es necesario transferir el programa a la tarjeta. Una vez que el proyecto ha sido compilado puede

ver la lista de todos los ficheros implicados pulsando el «+» que hay a la izquierda del nombre del fichero main.cpp.

Este proyecto está configurado para trabajar con el depurador (*debugger*), es decir, directamente sobre el *target* pero siendo este controlado desde el *host*, de modo que pueden consultarse y modificarse el valor de registros, variables, posiciones de memoria y emplear *breakpoints* (a diferencia de en los ejercicios anteriores, donde todo este proceso era *simulado* en el *host*, pero no existía un *target* físico).

La configuración de las opciones de depuración se realiza desde el menú Options for Target  dentro de la pestaña Debug, como se ve en la figura 13. Observe en esta figura cómo la opción Use Simulator aparece desmarcada y, en su lugar, aparece marcada la opción Use: ST-Link Debugger, que es el tipo de *debugger* incluido en las placas STM Nucleo.

Asegúrese de que todas las opciones en la parte derecha de esta pestaña se corresponden con las vistas en la figura 13.

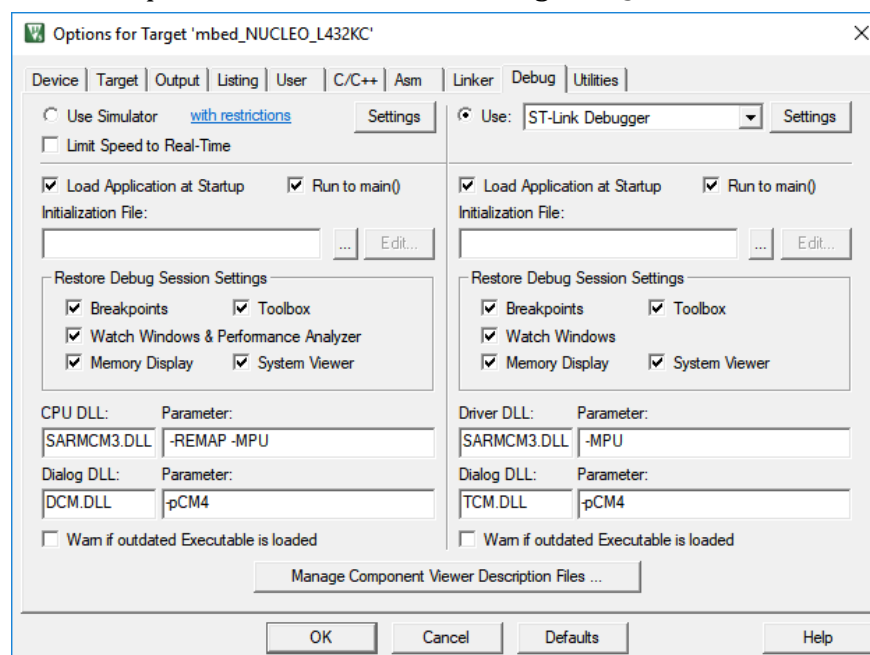


FIGURA 13: configuración de depuración para la placa mbed_NUCLEO_L432KC.

También deben ajustarse algunos parámetros del *debugger* accesibles desde Settings → Debug de la pestaña Debug. Dichos parámetros se encuentran descritos en la figura 14, destacados en rojo. Estos ajustes básicamente determinan qué interfaz concreta de depuración emplear (la interfaz SWD en vez de la JTAG), su frecuencia máxima de funciona-

miento (en función del *hardware* disponible en la placa) y ciertas opciones de conexión e inicialización de la interfaz del *debugger*. Verifique que dichos ajustes son como los mostrados en la figura 14.

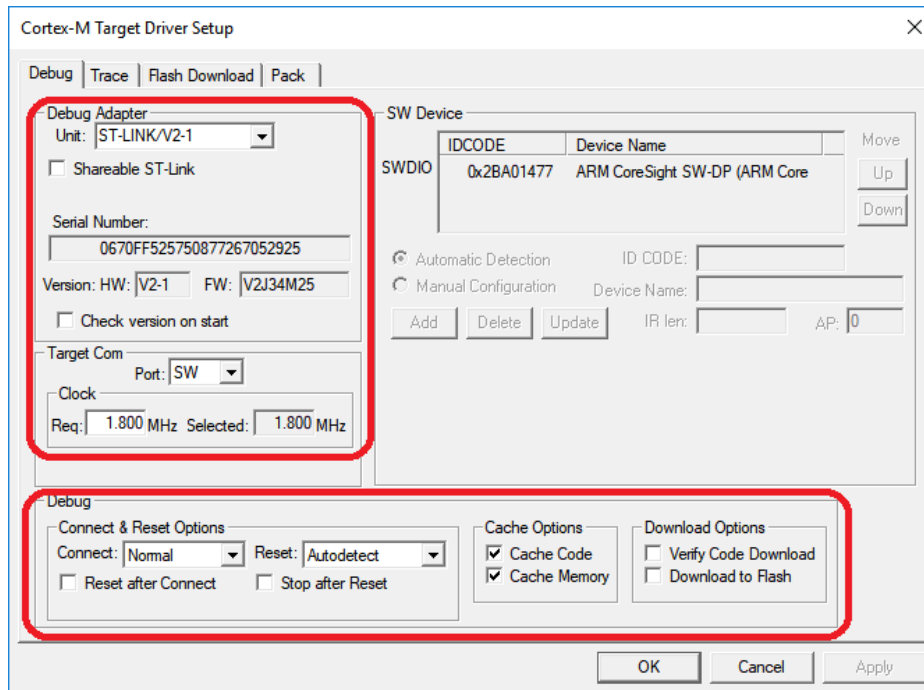


FIGURA 14: ajustes del *debugger* para la placa mbed_NUCLEO_L432KC.

En la misma ventana, pero en la pestaña *Trace*, debe ajustarse el valor de Core Clock a 80 MHz, como se ve en la figura 15, marcando además la casilla *Use Core Clock*. Esto permitirá al depurador medir correctamente el tiempo de ejecución de la aplicación, tiempo que, durante la depuración, se reporta en la línea de estado de la herramienta (la línea inferior de la ventana de *Keil μVision 5*, campo *t1*) y en la ventana de *Registers*, dentro del epígrafe *Internal*, campo *Sec*.

Por otro lado, también es necesario configurar el método empleado para volcar el ejecutable en la memoria *Flash* del microcontrolador. Para ello, en la pestaña *Utilities of Options for Target*, asegúrese de que está seleccionado *Use Target Driver for Flash Programming* y active las opciones *Use Debug Driver* y *Update Target before Debugging*, como ve en la figura 16. Además, dentro del botón *Settings* de esa misma pestaña, y en la pestaña *Flash Download* debe seleccionarse el algoritmo concreto de programación de la memoria *Flash* que emplea el procesador que se esté usando, algoritmo que es *STM32L4xx 256 KB Flash*.

MICROPROCESADORES

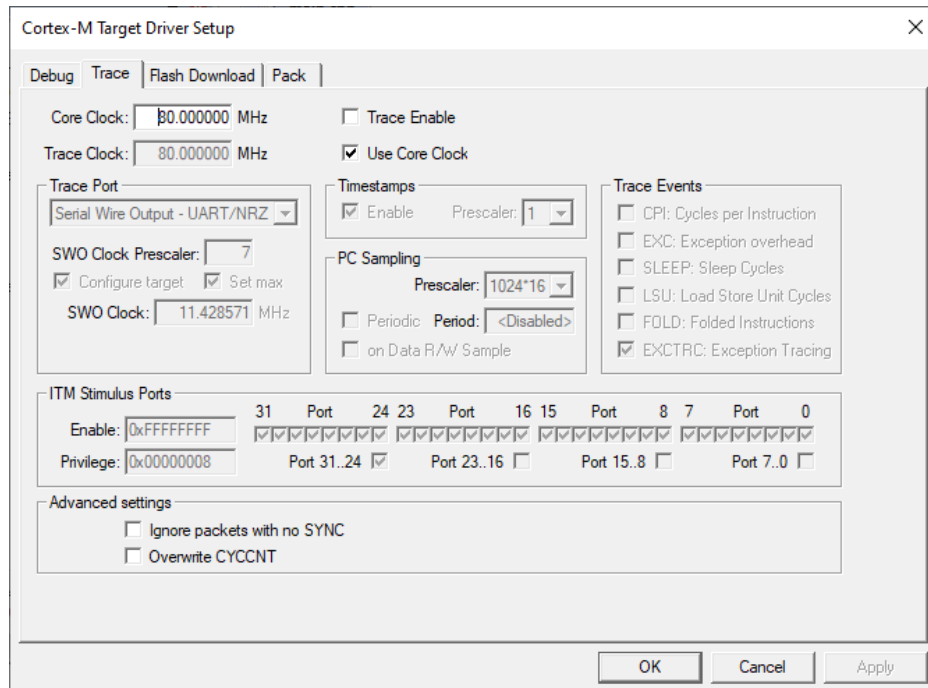


FIGURA 15: configuración, para el depurador, de la frecuencia de reloj del procesador.

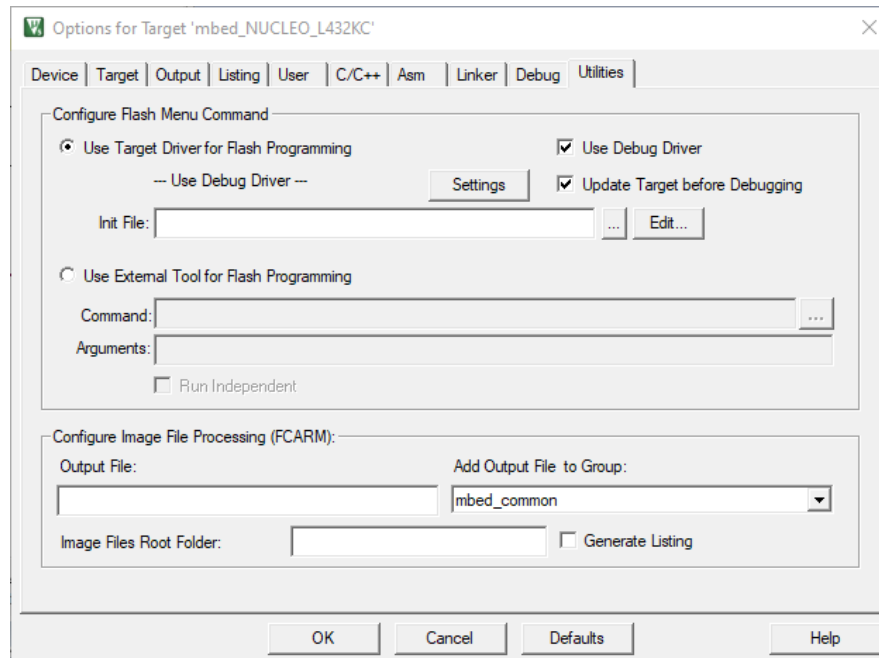


FIGURA 16: configuración del mecanismo de volcado a la memoria *Flash* en Keil μ Vision 5.

Asegúrese de que el algoritmo empleado (dentro del marco Programming Algorithm) es el recién descrito y de que existe una y solo una instancia del algoritmo. Si la configuración no fuera la deseada, emplee los botones Add y Remove para eliminar los algoritmos incorrectos y añadir los adecuados. En la figura 17 se muestran estos ajustes. Revise estos ajustes si la herramienta *Keil μ Vision 5* le informara de errores al volcar el programa en la placa.

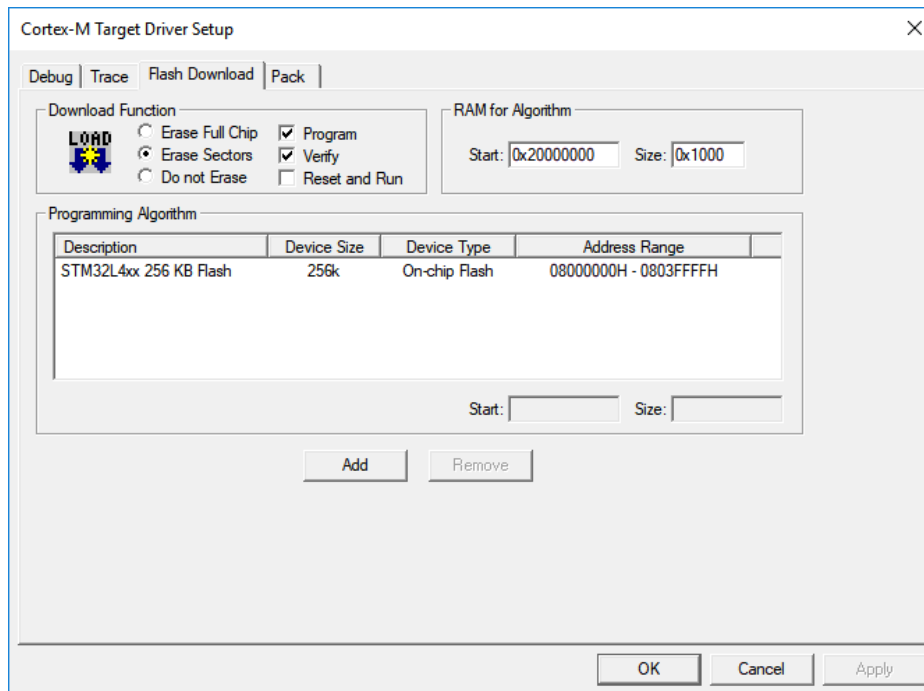















FIGURA 17: ajustes del algoritmo de programación de la *Flash* para la placa *mbed_NUCLEO_L432KC*.

Seguidamente ya es posible transferir el programa a la tarjeta. Para ello se debe pasar al modo de depuración pulsando el botón . Esto hará que el código se transfiera a la placa y comience a ejecutarse (el mismo botón  sirve para salir del modo de depuración para, por ejemplo, modificar y recompilar el programa). El proyecto está configurado para que automáticamente se ejecute el código hasta llegar a `main()`. A partir de este momento se puede (como se hizo en ejercicios anteriores):

- poner *breakpoints* mediante el botón  (haciendo *click* a la izquierda del número de línea);
- ejecutar el programa directamente empleando el botón ;

- ejecutar paso a paso utilizando los botones    —el primero de estos tres hace que, si lo que se trata de ejecutar es una función, se pase a ejecutar paso a paso las líneas dentro de la función; el segundo hace que, si se trata de ejecutar una función, esta se ejecute como si fuera una única instrucción; finalmente el tercero de los botones hace que se ejecuten todas las instrucciones hasta salir de la función en la que se encuentra el programa—;
- detener la ejecución en curso mediante el botón ;
- *resetear* el procesador mediante el botón .

Según se ejecuta paso a paso, la flecha que aparece a la izquierda del código  (o  en la ventana del desensamblador) se irá desplazando línea a línea. Tenga en cuenta que se puede ejecutar paso a paso en la ventana del código en C/C++ o del código en ensamblador según la que tenga en foco en cada momento. Para esta práctica ejecute el programa con el foco en la ventana de C/C++.

El botón  permite descargar la aplicación sobre el microcontrolador sin pasar el entorno *Keil μ Vision 5* a modo de depuración. Para que el programa, una vez descargado con , comience a ejecutarse deberá pulsar el botón de *reset* de la placa del microcontrolador, o bien desconecte y vuelva a conectar la alimentación de la placa.

Cuando el programa está parado es posible consultar sus variables (y modificar sus valores) y el contenido de la memoria (que también puede modificarse durante la depuración). Para ello es necesario acceder a las opciones Watch Windows y Watch Memory del menú View. Ambas ventanas se muestran en la parte inferior derecha de la pantalla. Consulte la ayuda de *Keil μ Vision 5* para más detalles. Tenga en cuenta que, para añadir una variable a una ventana de Watch debe hacerse *click* sobre el campo <Enter expression> de la ventana de Watch y escribir en él el *nombre completo* de la variable que se desea inspeccionar o modificar, que es una expresión de la forma:

`\nombre_de_fichero\nombre_de_función\variable`

y que identifica completamente a la variable. Si la variable fuese un objeto global su nombre completo sería:

`\nombre_de_fichero\variable`

En la figura 18 se muestra como añadir un Watch para la variable `tmp`.

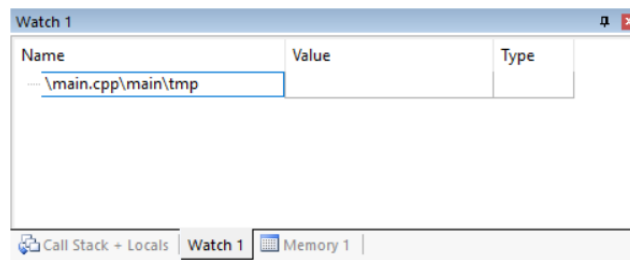






FIGURA 18: añadiendo una variable, mediante su nombre completo, a una ventana de Watch en Keil μ Vision 5.

Para este primer ejemplo:

1. compile el programa;
2. pase a modo de depuración;
3. lance la ejecución continua () , observe los LED;
4. ponga un punto de ruptura en la línea 20 (`g_ld1 = g_ldm;`);
5. una vez se detenga en el punto de ruptura, observe el estado de los LED en la placa y —mediante un *watch*— el valor de la variable `tmp`;
6. ejecute paso a paso con  —observando la evolución de los LED y el valor de `tmp`— hasta la línea 23 (`} // if (gb_tick_1s_evnt)`). En este momento lance la ejecución continuada de nuevo (botón ) hasta que vuelva a alcanzarse el *breakpoint*. Repita este proceso cuanto sea necesario hasta tener clara la evolución de los LED y el valor de `tmp`. No se preocupe, de momento, por el objeto `gb_tick_1s_evnt`, le es suficiente por ahora con saber que tal objeto es un *booleano* que se pone automáticamente a **true** una vez por segundo. Tampoco se preocupe, de momento, por la llamada a la función `sw_tick_serial_init()`.
7. Para terminar, modificando adecuadamente —mediante un *watch*— el valor de la variable `tmp` (sin modificar el programa) consiga que durante la ejecución continua del programa (botón ) se vean *dos* LED encendidos que se desplazan a la izquierda.

Tenga en cuenta que el mecanismo de *watch* solo puede acceder, y modificar, el contenido de una variable cuando esta se encuentra almacenada en memoria. Sin embargo podría ocurrir que el compilador determinase, buscando un ejecutable más eficiente o rápido, almacenar

dicha variable no en memoria, sino directamente en un registro del procesador. En tal caso no sería posible acceder al contenido de dicha variable —tanto en escritura como en lectura— mediante un *watch*. Por ello se ha añadido el calificativo **volatile** a la definición de la variable `tmp`:

```
bool volatile tmp = g_ld1;
```

Dicho calificativo fuerza al compilador a no *optimizar* el uso de dicha variable en forma alguna, con lo que la variable se almacenará siempre en memoria (y cada vez que se lea su valor, se leerá de memoria, y cada vez que se modifique su valor, se escribirá el nuevo valor en memoria) y podrá ser accedida, durante la depuración, mediante un *watch*. Recuerde esto si alguna vez el mecanismo de *watch* le informa de que no es posible acceder a alguna variable para mostrar su valor o modificarla. Eso sí, recuerde también que no tiene sentido, y por tanto no es posible, que mediante un *watch* u otro mecanismo similar se acceda a una variable que se encuentra fuera de alcance (*scope*). El calificativo **volatile** tiene otros usos que explorará más adelante.

A partir de este instante es necesario que siempre que se ejecute un programa se realice aplicando el método de depuración que se ha explicado para el programa de ejemplo. Sepa que en los exámenes de laboratorio se evaluará el manejo que hace usted de los recursos de depuración, *breakpoints* y *watches* incluidos, por lo que su uso a lo largo de las prácticas no es solo recomendable, sino imprescindible.

Para terminar este primer apartado, deberá modificar el programa de modo que los tres objetos `DigitalOut` de la librería *mbed* sean sustituidos por un único objeto de la clase `BusOut`, según:

```

#include "mbed.h"
#include "pinout.h"
#include "sw_tick_serial.h"

// leds, when in a int8_t, they are LMR
static BusOut      g_leds(LDR_PIN, LDM_PIN, LDL_PIN);

int main (void) {
    sw_tick_serial_init();
    g_leds = 1;
    for (;;) {
        if (gb_tick_1s_evnt) {
            gb_tick_1s_evnt = false;
            g_leds = ((4 == g_leds) ? 1 : (g_leds << 1));
        } // if (gb_tick_1s_evnt)
    } // forever
} // main()

```

Compile este programa y verifique su funcionamiento sobre la placa. Si desconoce los operadores de desplazamiento (<< y >>) o el operador ternario (? :) del lenguaje C/C++, puede consultar la *web*, le serán de utilidad más adelante.

5.10. Ejercicio 4 - Eventos

Para comprender completamente el funcionamiento del programa deberá conocer el funcionamiento de la librería *sw_tick_serial* que se emplea. Abra el fichero *sw_tick_serial.h* y lea los comentarios acerca del objeto *gb_tick_1s_evnt*.

En los sistemas basados en microprocesador que interactúan con el exterior existe el concepto de *evento* (en inglés *event*). Un evento es cualquier suceso susceptible de generar una reacción por parte del sistema. En este caso, en el que el sistema debe apagar un LED y encender otro una vez por segundo, el evento que provoca esa reacción es el paso de un segundo.

En estos sistemas existe, para cada evento, un *flag* (o *indicador* o *bandera*) que indica su ocurrencia (se activa a *true* cada vez que el evento sucede). En este ejemplo el *flag de evento* es el objeto *booleano*

`gb_tick_1s_evnt`, que se pone a **true** una vez por segundo. El programa, cada vez que este *flag* se activa, realiza la actualización de los LED y desactiva el *flag* a **false**, que volverá a activarse automáticamente al cabo de un segundo. El mecanismo por el cual este *flag* de evento se activa periódicamente será el objeto de la práctica 2 de este laboratorio, por el momento solo necesita saber que para arrancar el mecanismo que genera todos los *flags* de evento que soporta la librería *sw_tick_serial* es necesario llamar, una sola vez y desde `main()`, a la función `sw_tick_serial_init()` de la misma.

Muchas veces los *flags* de evento se denominan, simplemente, eventos, lo que puede resultar algo confuso, pero debe tener cuidado para saber distinguir lo que es un evento de lo que es un *flag* de evento.

La librería *sw_tick_serial* que le proporcionamos soporta varios eventos diferentes, los que son de interés para esta sesión de la práctica son los señalizados mediante los *flags* de evento:

```
extern bool volatile gb_tick_1ms_evnt;
extern bool volatile gb_tick_10ms_evnt;
extern bool volatile gb_tick_100ms_evnt;
extern bool volatile gb_tick_1s_evnt;
extern bool volatile gb_tick_10s_evnt;
```

Todos ellos son *booleanos* que se ponen periódicamente a **true** con el periodo indicado en su nombre.

Copie el proyecto completo de *Keil μVision 5* del apartado anterior (los contenidos de la carpeta `MICR\P1\S1\E3`) sobre la carpeta `MICR\P1\S1\E4` y trabaje sobre este nuevo proyecto. Modifique el programa del apartado anterior (el que emplea un `BusOut`) para que los LED se actualicen a una frecuencia de 10 Hz. Verifique el funcionamiento sobre la placa.

5.11. Ejercicio 5 - Control del *display* de 7 segmentos



En este apartado se pide que escriba un programa que muestre, en el *display* de 7 segmentos, la cuenta decimal ascendente módulo 10. La cuenta debe actualizarse una vez por segundo. En la carpeta `MICR\P1\S1\E5` encontrará un fichero ejecutable `.bin` con la aplicación ya compilada, de modo que pueda comprobar cuál es el funcionamiento

esperado cargándola directamente en la placa, según se describió en el apartado 5.5.3 en la página 18.

En la misma carpeta MICR\P1\S1\E5 encontrará también un proyecto de *Keil* con una aplicación vacía sobre la que trabajar para resolver este apartado. El proyecto incluido en esta carpeta ya contiene la librería *mbed* lista para ser usada y el entorno *Keil μ Vision 5* preparado para trabajar sobre la placa STM.

5.11.1. ADICIÓN DE GROUPS, FICHEROS Y LIBRERÍAS A UN PROYECTO

Para realizar este apartado podrá usar la librería *sw_tick_serial* que se ha descrito anteriormente y que encontrará en la carpeta MICR\sw_tick_serial. Dicha librería está formada por dos ficheros: un .h (cabecera) y un .lib (objeto). Para incluir estos ficheros en el proyecto de *Keil μ Vision 5* se procederá de la siguiente forma:

1. Copie el fichero *sw_tick_serial.h* de su ubicación original (MICR\sw_tick_serial) a la carpeta en la que está el proyecto de *Keil μ Vision 5* (MICR\P1\S1\E5). Esto permite incluirla simplemente con `#include "sw_tick_serial.h"` sin tener que especificar su ruta.
2. Pulsando sobre el icono Manage Project Items...  se abrirá un diálogo en el que se escogerá la pestaña Project Items, como se ve en la figura 19.
3. Para incluir este fichero dentro del *group* C++ Sources seleccione este *group* en el cuadro del medio y pulse el botón Add Files..., se abrirá el diálogo de inclusión de ficheros al *group*, como se ve en la figura 20. En el campo Files of Type: seleccione Text File (*.txt; *.h; *.inc) —dado que el fichero que quiere añadirse al proyecto tiene extensión .h—. Seleccione el fichero *sw_tick_serial.h* y pulse Add y luego Close. Verá como el fichero *sw_tick_serial.h* queda añadido al *group* C++ Sources.
4. El fichero *sw_tick_serial.h* contiene la declaración de los objetos que forman la librería *sw_tick_serial*, pero no contiene ninguna funcionalidad, esta se encuentra dentro del fichero .lib (objeto), que también deben ser incluidos en el proyecto. En este caso se va a crear un *group* específico para él que se llamará *sw_tick_serial*. En la misma pestaña Project Items pulse sobre el botón de crear nuevo *group*  (en la parte central), dé nombre *sw_tick_serial* al

group y empleando los iconos   ubique dicho *group* por encima del *group* C++ Sources.

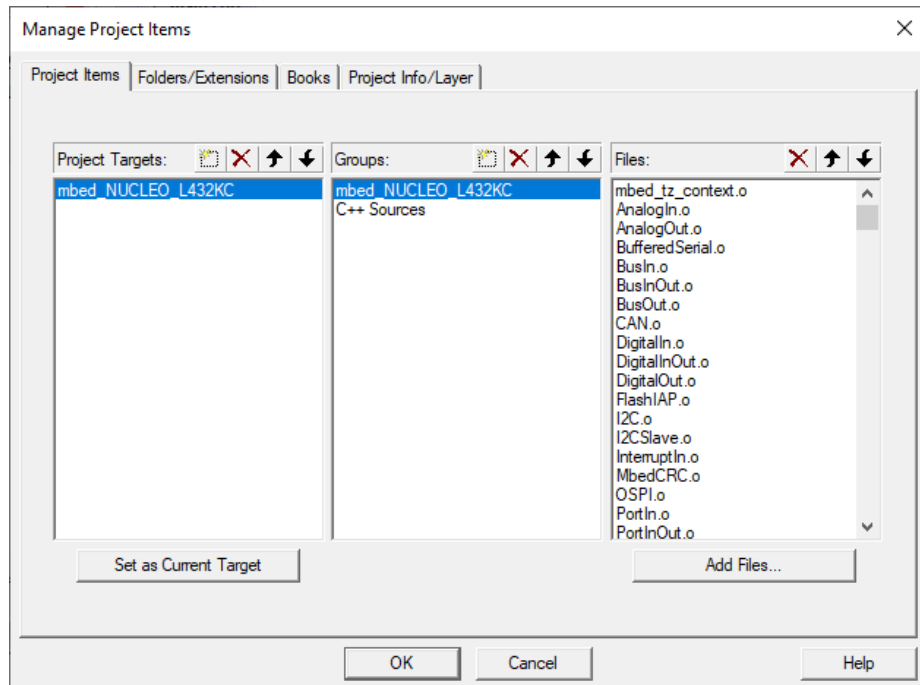


FIGURA 19: pestaña Project Items.

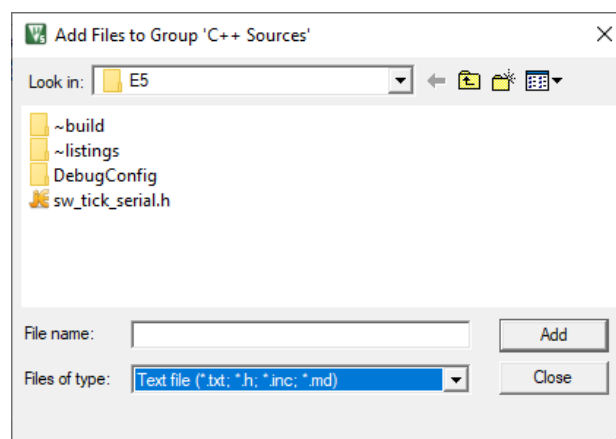


FIGURA 20: diálogo Add Files to Group.

5. Tal como hizo en el punto 3 anterior, añada ahora el fichero `.lib` en `MICR\sw_tick_serial` al *group* `sw_tick_serial` recién creado. Ahora debe elegir archivos del tipo Library File (`*.lib`). Todo esto añade el fichero objeto de la librería `sw_tick_serial` al proyecto dentro de un *group* llamado `sw_tick_serial`, como se aprecia en la figura 21.

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

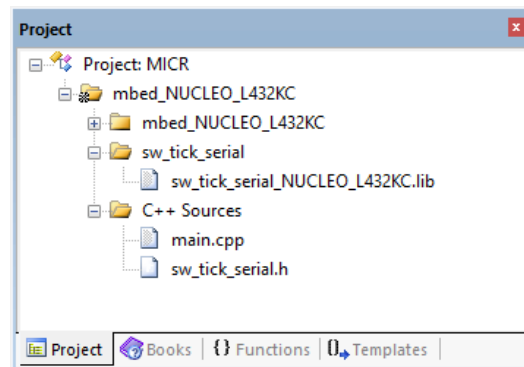


FIGURA 21: *group* y archivos de *sw_tick_serial* añadidos al proyecto.

Observe que, añadiendo la librería *sw_tick_serial* de esta forma, esta librería *no es copiada* desde su ubicación original a la ubicación del nuevo proyecto (solo se ha copiado el fichero *.h*), lo que minimiza el espacio ocupado en el disco para el caso de que una librería vaya a ser reutilizada muchas veces (y sea grande, como ocurre con la librería *mbed*). Sin embargo es imperativo mantener la ubicación relativa de la librería y del proyecto que la usa en el disco, de otra manera *Keil μ Vision 5* no podría localizar la librería en el disco. Por eso en la sección 4 de la página 7 se insistió en que todos los proyectos de *Keil μ Vision 5* de esta asignatura deben ubicarse 3 niveles de carpetas por debajo de la carpeta *MICR*, para asegurar que siempre se pueden encontrar todas las librerías requeridas.

Empleando estos mismos mecanismos puede incluir también en el proyecto (dentro del *group* *C++ Sources*) el fichero *pinout.h* que vio en apartados anteriores y que puede serle de utilidad.

5.1.1.2. REQUISITOS PARA ESTA APLICACIÓN

Las 7 señales SGA a SGG que controlan los segmentos del *display* se agruparán en un *BusOut* (siendo SGA el LSB). El programa deberá disponer de una variable que mantenga el estado de la cuenta (un número del 0 al 9 que se incrementará una vez por segundo) y de una función capaz de traducir ese valor de cuenta en el código necesario a asignar al *BusOut* para que en el *display* se muestre el símbolo asociado (ver figura 6 en la página 16). El prototipo de dicha función sería:

```
int8_t to_7seg(uint8_t code);
```

donde `code` es el valor de la cuenta y el valor devuelto por la función es el que hay que asignar al `BusOut`. Para implementar esta función se recomienda emplear una *tabla de búsqueda* (*lookup table*). Se trata de un *array* de datos en los que cada posición representa un código diferente, de modo que si se indexa el *array* con el valor 3, el código que se encuentra en esa posición debe ser el que hay que asignar al `BusOut` para que se represente el valor 3 en el *display*. Este *array* puede declararse como:

```
static int8_t const sseg[] = {
    <segs_del_0>, <segs_del_1>,
    <segs_del_2>, <segs_del_3>,
    ...,
    <segs_del_8>, <segs_del_9>
};
```

donde las constantes `<segs_del_0>` a `<segs_del_9>` debe calcularlas para que, para cada valor de `code`, `sseg[code]` indique el valor (apagado o encendido) que deben tomar los segmentos del *display* para representar el valor `code`. Asegúrese de que su función `to_7seg()` se comporta adecuadamente ante valores inválidos del parámetro de entrada. En particular, si el parámetro de entrada está fuera del rango permitido (de 0 a 9), la función retornará el valor necesario para que se apaguen todos los segmentos del *display*. Tenga en cuenta que esta función no debe acceder directamente al objeto de la clase `BusOut` para darle valor, solo retorna el valor que el programa principal debe asignar a ese objeto para mostrar la cifra deseada.

Verifique sobre la placa que su programa reproduce exactamente el funcionamiento previsto.

5.12. Ejercicio 6 - Gestión de varios eventos

En este apartado se pide que escriba un programa que combine la funcionalidad de los dos programas anteriores, es decir, que muestre en el *display* de 7 segmentos la cuenta decimal ascendente de módulo 10 a una frecuencia de 1 Hz y que, simultáneamente, vaya «desplazando» a izquierdas un LED a una frecuencia de 10 Hz.

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

Se recomienda que copie proyecto del apartado anterior en MICR\P1\S1\E5 (cuidado, no incluya el .bin que se suministró) dentro de la carpeta MICR\P1\S1\E6 y que realice este apartado en esta carpeta. En la misma carpeta MICR\P1\S1\E6 encontrará un fichero ejecutable .bin que puede emplear para observar el funcionamiento esperado para este apartado.

Terminan aquí las tareas a realizar previas a la primera sesión presencial de esta práctica.

6. TRABAJOS PREVIOS A LA SEGUNDA SESIÓN PRESENCIAL

Se describen a continuación los trabajos a realizar antes de la segunda y última sesión presencial de la práctica.

6.1. Ejercicio 7 - Montaje del circuito

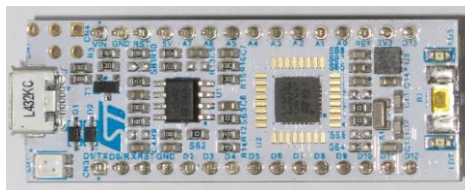
Se continúa el montaje del circuito añadiendo ahora los recursos necesarios para la parte restante de esta práctica.

6.1.1. MONTAJE DE PULSADORES Y SEGUNDO *DISPLAY*

El circuito necesario para la segunda sesión de esta práctica añade al circuito de la sesión anterior: 3 pulsadores; otro *display* de 7 segmentos y la correspondiente circuitería de multiplexado de los *displays*.

Cada uno de los pulsadores gobernará una de las señales SWL, SWM y SWR (de *SWitch Left*, *SWitch Middle* y *SWitch Right*). Los pulsadores entregarán un nivel lógico *bajo* al pulsarse. A su vez, las señales SWL, SWM y SWR se conectarán a los siguientes pines del microcontrolador:

VIN	GND	NRST	+5V	A7	A6	A5	A4	A3	A2	A1	A0	AREF	+3V3	D13
-----	-----	------	-----	----	----	----	----	----	----	----	----	------	------	-----



D1	D0	NRST	GND	D2	D3	D4	D5	D6			D9	D10	D11	D12
----	----	------	-----	----	----	----	----	----	--	--	----	-----	-----	-----

SWL SWM

SWR

Como ya conoce (por las asignaturas Electrónica I y Electrónica II) debe incluirse una resistencia de *pull-up* para cada pulsador, sin embargo en este laboratorio se emplearán las resistencias internas de *pull-up* de los microcontroladores, por lo que no será necesario montarlas.

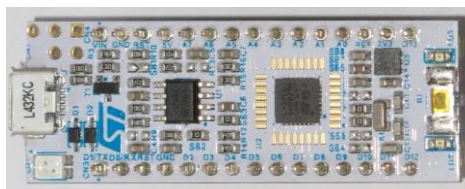
Por lo que respecta al segundo de los *displays* de 7 segmentos, este se montará de forma que comparta las resistencias de limitación con el *display* ya existente (en total habrá 14 segmentos, pero solo 7 resistencias de limitación de corriente) y también compartirán las señales SGA a

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

SGG. Sin embargo, los cátodos comunes de ambos *displays* se controlarán mediante transistores, de modo que, en cada instante del tiempo, solo uno de los cátodos comunes tenga un camino de baja impedancia a masa, de manera que solo pueda estar encendido uno de los *displays* al mismo tiempo. Este tipo de montaje se denomina *multiplexación de displays* y permite controlar *displays* de varias cifras con un número pequeño de señales (tantas como segmentos tenga un *display* más el número de *displays*). Puede encontrar los detalles de esta conexión en el apartado 3.6.3 del libro de Toulson y Wilmshurst, en las clases de teoría o también en la *web*. Sin embargo debe tener en cuenta que en este laboratorio los transistores para la multiplexación del *display* serán bipolares NPN (en el libro citado son MOSFET de acumulación canal N) y, concretamente, serán de los tipos BC547 o 2N3904, a su elección. Cada transistor será controlado por una de las señales DSL y DSR (de *DiSplay Left* y *DiSplay Right*), de modo que cuando una de estas señales se active a nivel *alto*, el *display* correspondiente se encienda, permaneciendo apagado en caso contrario. Las señales DSL y DSR se conectarán a los siguientes pines del microcontrolador:

DSL DSR

VIN	GND	NRST	+5V	A7	A6	A5	A4	A3	A2	A1	A0	AREF	+3V3	D13
-----	-----	------	-----	----	----	----	----	----	----	----	----	------	------	-----



D1	D0	NRST	GND	D2	D3	D4	D5	D6			D9	D10	D11	D12
----	----	------	-----	----	----	----	----	----	--	--	----	-----	-----	-----

Como ya conoce (por la asignatura Electrónica I), será necesario polarizar cada transistor adecuadamente para conseguir que se saturen o corten en función del nivel lógico presente en la señal DSL/DSR que lo controle. Para ello deberá calcular el valor de las resistencias de base que deben emplearse en dicho montaje. Los parámetros necesarios para este cálculo se encontrarán en la *datasheet* de los microcontroladores (disponibles en *Moodle*) y del transistor concreto que haya empleado (cuya *datasheet* debe localizar usted).

6.1.2. VERIFICACIÓN DEL MONTAJE Y CARGA DE SOFTWARE

En este apartado se cargará una aplicación de ejemplo, ya compilada, que pretende simplemente verificar que el conexionado de los recursos anteriores es correcto. Para ello, una vez montados los pulsadores y el segundo *display* de 7 segmentos (con su circuitería de multiplexación) junto con el microcontrolador en las placas de prototipado, se volcará sobre la placa el ejecutable que se adjunta (carpeta MICR\P1\S2\E7). Este programa presenta un número del 00 al 99 en el *display*. Si se pulsa el botón derecho, el número se incrementa en una unidad; si se pulsa el botón izquierdo, se decrementa en una unidad; finalmente, si se pulsa el botón central, el número mostrado (n) pasa a ser $99 - n$. A la vez, los LED mostrarán, a una cadencia de un segundo, la cuenta binaria ascendente de 3 bits.

6.2. Ejercicio 8 - Gestión de los interruptores

Deberá elaborar un programa que muestre, en el *display* de 7 segmentos de la derecha, la cuenta decimal ascendente de módulo 10, incrementándose la cuenta exclusivamente con cada pulsación del pulsador izquierdo. A la vez, encenderá uno de los tres LED que se encuentran en la placa e irá «desplazando» el LED encendido, hacia la izquierda, una posición por cada pulsación del pulsador derecho.

Según lo dicho en la sección 6.1, para conseguir encender el *display* derecho y apagar el izquierdo la señal DSR deberá permanecer a nivel alto y la señal DSL a nivel bajo.

Para saber cuándo se activan los pulsadores debe tener en cuenta que la librería *sw_tick_serial* que empleó en la anterior sesión provee los *flags* de evento (consulte *sw_tick_serial.h*):

```
extern bool volatile gb_sw1_evnt;
extern bool volatile gb_swm_evnt;
extern bool volatile gb_swr_evnt;
```

que se ponen a **true** cada vez que se detecta un flanco de bajada en los puertos conectados a las señales SWL, SWM y SMR, respectivamente. Es decir, estos *flags* se activan con cada pulsación del pulsador correspondiente.

Se recomienda que para la realización de este apartado se parta del programa que se elaboró al final de la sesión anterior, modificándolo

convenientemente. Copie aquel proyecto de *Keil μ Vision 5* dentro de la carpeta MICR\P1\S2\E8 y trabaje sobre esta copia.

6.3. Ejercicio 9 - Multiplexación «lenta» de *displays*

Modifique ahora el programa anterior para que el *display* usado para presentar la cuenta se elija empleando el interruptor restante. Inicialmente el *display* empleado será el derecho, pero cada vez que se pulse el interruptor central, el *display* empleado conmutará *inmediatamente* al otro. Copie el proyecto completo del programa anterior dentro de la carpeta MICR\P1\S2\E9 y trabaje sobre esta copia. Dentro de esta misma carpeta encontrará un ejecutable de esta aplicación que puede emplear para contrastar su funcionamiento con el de su programa.

6.4. Ejercicio 10 - Trabajo con varios ficheros fuente

Una vez que esté operativo el programa anterior, lo dividirá en dos ficheros de código independientes (dos ficheros con extensión .cpp) que ayuden a organizar el código desarrollado en base a su funcionalidad. Copie el proyecto de *Keil μ Vision 5* del apartado anterior (sin incluir los ejecutables .bin que se suministraron) dentro de la carpeta MICR\P1\S2\E10 y trabaje sobre esta nueva copia. En uno de los ficheros .cpp se incluirá exclusivamente la función empleada para la conversión a siete segmentos —`to_7seg()`— (fichero `to_7seg.cpp`), mientras que el otro contendrá el código restante —incluyendo `main()`, fichero `main.cpp`—.

La función `to_7seg()` se modificará también para que, además de los dígitos del 0 al 9, muestre los 16 símbolos que se presentaron en la figura 6 de la página 16 (para valores del parámetro de entrada desde 0 hasta 15). Como antes, para valores inválidos del parámetro de entrada la función retornará el código necesario para apagar todos los segmentos.

Para realizar la separación en varios ficheros fuente debe tener en cuenta lo explicado en la asignatura Programación I respecto al funcionamiento de los ficheros de cabecera (.h) y las diferencias entre *declarar* y *definir* un objeto en C/C++. Las funciones se declaran mediante su *prototipo*, mientras que las variables (globales) requieren el uso del

modificador `extern` para ser declaradas. El estudio del fichero `sw_tick_serial.h` puede serle de utilidad en este sentido. Deberá también recordar cómo añadir ficheros al proyecto de *Keil μ Vision 5* (ver la sección 5.11.1), de modo que tanto `to_7seg.h` como `to_7seg.cpp` estén dentro de un nuevo *group* llamado `to_7seg`.

Por otro lado, debe considerar que los tipos de datos `int8_t` y `uint8_t` usados por `to_7seg()` se definen dentro de la librería *mbed*, por ello, `mbed.h` deberá incluirse tanto en `to_7seg.cpp` (obviamente) como en `to_7seg.h` —puesto que el prototipo de la función `to_7seg()` también requiere esos tipos—. Sin embargo cuando, a su vez, se incluya `to_7seg.h` en `main.cpp` puede ocurrir que *mbed* se incluya en `main.cpp` dos veces (explícitamente con `#include "mbed.h"` y de forma menos evidente con `#include "to_7seg.h"`). Esta es una situación que debe evitarse, los ficheros de cabeceras deben incluirse solo si no han sido incluidos por anterioridad. Para solventar estas situaciones suele hacerse uso de las directivas del preprocesador: `#define`, `#ifdef`, `#ifndef` y `#endif`, todas ellas seguidas, excepto `#endif`, por un NOMBRE_DE_MACRO. Su utilidad es:

- con `#define` NOMBRE_DE_MACRO se define (crea) un *macro* (vacío) de nombre NOMBRE_DE_MACRO;
- el código de un fichero fuente comprendido entre `#ifdef` NOMBRE_DE_MACRO y `#endif` no será compilado si el macro (o *símbolo*) NOMBRE_DE_MACRO no ha sido definido anteriormente en ese fichero fuente;
- el código de un fichero fuente comprendido entre `#ifndef` NOMBRE_DE_MACRO y `#endif` no será compilado si el símbolo NOMBRE_DE_MACRO ha sido definido anteriormente en ese fichero fuente.

Ahora cada fichero de cabeceras `.h` (por ejemplo, `sw_tick_serial.h`) tiene la estructura:

```
#ifndef SW_TICK_SERIAL_H
# define SW_TICK_SERIAL_H
# include "mbed.h"
...    // resto de contenidos
#endif // SW_TICK_SERIAL_H
```

Si un fichero fuente intentase incluir varias veces a este fichero (directa o indirectamente), en la primera de las inclusiones el símbolo `SW_TICK_SERIAL_H` no estaría definido, con lo que `#ifndef` incluiría el resto del fichero (lo que, además, definiría el macro `SW_TICK_SERIAL_H`

a través del `#define`). Sin embargo, la segunda —y sucesivas— inclusiones del mismo no tendrían efecto, ya que en ellas el macro `SW_TICK_SERIAL_H` ya se encontraría definido (desde la primera inclusión) y el `#ifndef` inicial evitaría que volviese a incluirse el cuerpo del fichero `.h`.

Deberá dotar a sus ficheros de los mecanismos necesarios para evitar la multiplicidad de inclusiones que acaba de comentarse. Puede usar el código de `sw_tick_serial.h` como ejemplo. El nombre de macro que debe usar es `TO_7SEG_H` (suele emplearse el nombre de fichero `.h` en mayúsculas y cambiando el punto —que es un carácter inválido para este uso— por un guion bajo).

A partir de este momento se espera que esta agrupación de funcionalidades en distintos ficheros fuente, mediante el uso de ficheros de cabecera, con el objeto de organizar el código se realice sin necesidad de advertirlo en los enunciados de las actividades.

6.5. Ejercicio 11 - Multiplexación de *displays*

En este apartado se pide que, empleando la librería `sw_tick_serial` (como se ha venido haciendo hasta ahora) escriba un programa que:

- muestre en el *display* de la derecha un número del 0 al 9 (inicialmente 0);
- incremente ese número con cada pulsación del botón central, si el número es 9 no se realizará el incremento;
- decremente ese número con cada pulsación del botón izquierdo, si el número es 0 no se realizará el decremento;
- ponga el número a 0 con cada pulsación del botón derecho;
- muestre el carácter «n» en el *display* izquierdo;
- encienda un LED y vaya «desplazando» ese LED a la izquierda a una frecuencia de 10 Hz.

Copie alguno de los proyectos de *Keil μ Vision 5* de las sesiones anteriores (el que considere más útil para este ejercicio) dentro de la carpeta `MICR\P1\S2\E11` y trabaje sobre esta copia para realizar este programa. En esa misma carpeta encontrará un ejecutable —MUX.bin— que le permitirá observar sobre la tarjeta el funcionamiento esperado. Encontrará también otro fichero ejecutable —Sombras.bin— que muestra un funcionamiento defectuoso del programa. En este caso el símbolo mostrado en un *display* se aprecia atenuado (como una sombra o imagen

fantasma) sobre el otro (las combinaciones «n1» y «n7» seguramente sean las más notables)*. Su programa debe evitar este tipo de comportamientos.

Para conseguir que en el *display* se muestren simultáneamente dos símbolos distintos debe recordar que, en el apartado 6.3, pudo mostrar un símbolo en un *display* o en el otro, cambiando de *display* con cada pulsación de un botón. Si ese cambio de *display*, en vez de hacerse al ritmo marcado por las pulsaciones, se hiciese periódicamente a una frecuencia suficientemente elevada, el ojo no apreciaría el parpadeo y vería el mismo símbolo a la vez en los dos *displays*. Si en estas condiciones el programa fuese tal que, cuando se encienda el *display* derecho muestre un símbolo (el número) y cuando se enciende el izquierdo otro símbolo (la «n») el ojo percibiría la combinación de las dos imágenes, formando el mensaje buscado.

La frecuencia a la que se cambia de *display* (*frecuencia de multiplexación*) debe ser tal que el ojo no aprecie el parpadeo. Una regla cómoda es fijar un valor *mínimo* para esa frecuencia igual 50 Hz multiplicado por el número de *displays* (o *campos*), en este caso, al haber dos *displays*, la frecuencia mínima de multiplexación sería de 100 Hz. A frecuencias próximas o inferiores a esta el ojo puede llegar a percibir el parpadeo del *display* (especialmente si el *display* se encuentra en movimiento dentro del campo de visión del observador).

La frecuencia de multiplexación máxima viene fijada por la capacidad de los transistores de conmutar correctamente a esa frecuencia y porque el procesador sea suficientemente rápido para realizar la gestión del *display* a esa velocidad.

Se propone que se emplee para este ejercicio una frecuencia de multiplexación de 1 kHz. Aunque puede parecer algo elevada, esta frecuencia es: sin duda superior a la frecuencia mínima; cómoda de obtener empleando la librería *sw_tick_serial*; no tan elevada como para que los transistores no puedan conmutar o el procesador no sea capaz de

* Estas sombras pueden ser más llamativas o no en función de los valores concretos que haya elegido para las resistencias de limitación de corriente de los segmentos y de base de los transistores, así como del transistor concreto que se haya empleado. También, a menor luz ambiente, más aparentes son. Igualmente, desmontando el *display* de la derecha podrá ver (con Sombras.bin) como se aprecia fácilmente el número que debiera presentarse en el *display* derecho, sobre la «n» del izquierdo.

realizar la gestión; pero suficientemente alta como para que puedan producirse sombras si existe una mala gestión del *display* por parte del programa*. En las siguientes prácticas, en las que ya no se hará uso de la librería *sw_tick_serial*, podrá emplear, si lo desea, frecuencias de multiplexación más bajas.

Deberá emplear para este apartado su función `to_7seg()`, manteniéndola en ficheros (.h y .cpp) separados, como en el apartado anterior.

6.6. EJERCICIO 12 - AAPCS

En este apartado se pide que reemplace el fichero `to_7seg.cpp` que ha venido empleando en los dos anteriores apartados por otro `to_7seg.s` que provea una función equivalente a `to_7seg()`, pero escrita en lenguaje de ensamble, obviamente siguiendo el AAPCS. Copie el proyecto de *Keil μ Vision 5* del apartado anterior (sin incluir los ejecutables .bin que se suministraron) dentro de la carpeta MICR\P1\S2\E12 y trabaje sobre esta nueva copia

Para añadir un nuevo fichero, dentro del *group* `to_7seg`, al proyecto de *Keil μ Vision 5*, pique con el botón derecho sobre el nombre del *group*, en la ventana izquierda Project y seleccione «Add New item to Group 'to_7seg'...», lo que abrirá el diálogo de adición de nuevos ficheros que puede ver en la figura 22. Seleccione el tipo del nuevo fichero como «Asm File (.s)», ingrese el nombre `to_7seg` y pulse Add, de esta manera se incorporará un nuevo archivo vacío `to_7seg.s` al proyecto en el *group*

* Existen diversos mecanismos que pueden dar lugar a la aparición de las sombras, siendo los más habituales:

- hay instantes del tiempo en los que ambos *displays* están encendidos simultáneamente;
- hay instantes del tiempo en los que el *display* que está encendido está mostrando información que no le corresponde a él.

Debe asegurarse de que su código, en ninguna circunstancia, da lugar a estos comportamientos. Revise para ello la secuencia temporal de las señales *DSL*, *DSR* y *SGX*. Si su código produjese sombras, el estudio con el osciloscopio de las señales *DSL* y *DSR* para los ejecutables *MUX.bin* y *Sombras.bin* y para su aplicación, le podría ser de utilidad para comprender la naturaleza del problema.

to_7seg, como se ve en la figura 23. Edite el fichero to_7seg.s añadiendo una función `to_7seg()`, pero ahora escrita en lenguaje de ensamble.

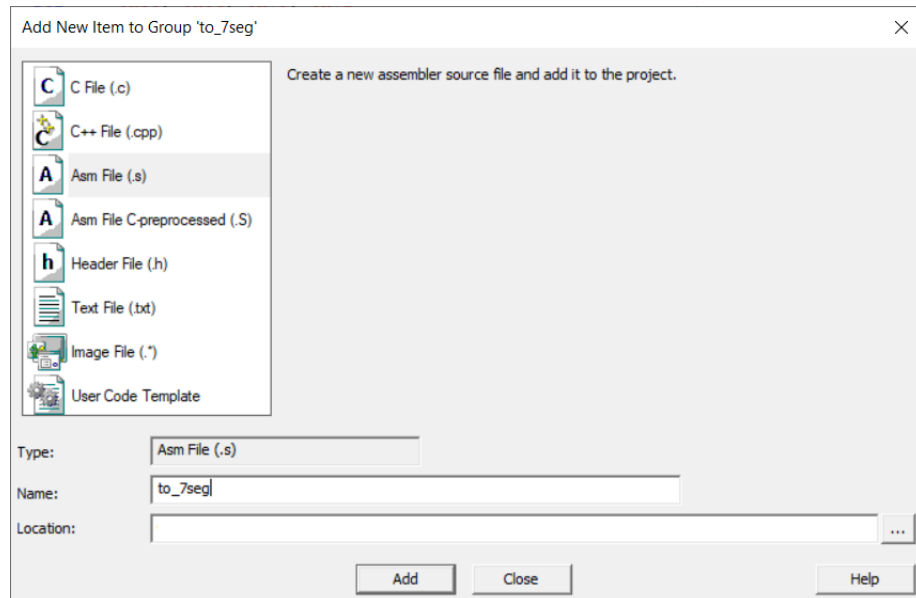


FIGURA 22: diálogo de adición de nuevos ficheros a un *group* en Keil μ Vision 5.

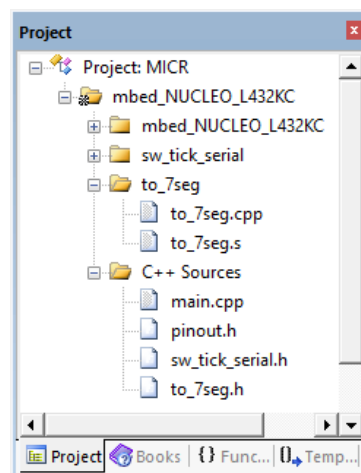


FIGURA 23: fichero to_7seg.s añadido al *group* to_7seg del proyecto.

Sin embargo, ahora mismo el proyecto dispone de dos funciones `to_7seg()`, la descrita en to_7seg.cpp y la nueva en to_7seg.s, pero solo debe compilarse y *linkarse* una de ellas, en este caso la de to_7seg.s. Para evitar la compilación del fichero to_7seg.cpp, de modo que la

función finalmente incorporada al ejecutable de la aplicación sea la descrita en `to_7seg.s`, pique con el botón derecho sobre el fichero que desea eliminarse del proceso de compilación `to_7seg.cpp` y seleccione la opción «Options for File 'to_7seg.cpp'...», como se ve en la figura 24.

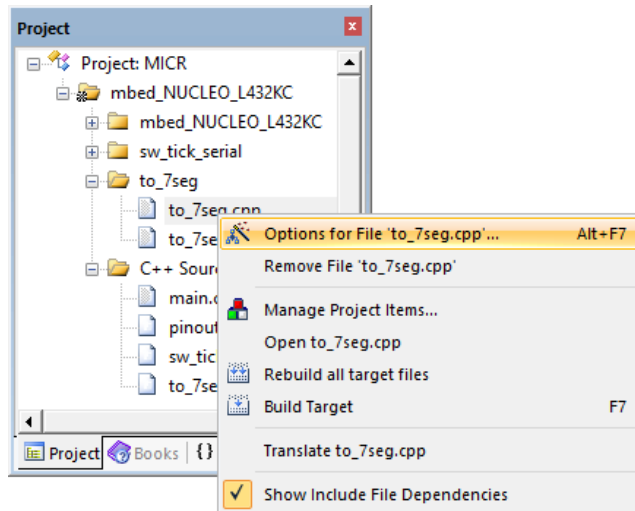


FIGURA 24: accediendo a las opciones del archivo `to_7seg.cpp`.

En el diálogo que aparece con las opciones del archivo `to_7seg.cpp` desmarque la opción «Include in Target Build», tal como se ve en la figura 25. De este modo el fichero `to_7seg.cpp` sigue formando parte del proyecto (lo que es de interés si, por ejemplo, desea volver a la versión en C++ de la función `to_7seg()` en vez de a la versión en lenguaje de ensamble) y es accesible desde el IDE (*Integrated Development Environment*) de *Keil μVision 5*, pero no es compilado ni interfiere con la función del mismo nombre descrita en el archivo `to_7seg.s`. El hecho de que un fichero se encuentre excluido de la compilación se muestra en la herramienta *Keil μVision 5* mediante un pequeño símbolo de prohibido junto al nombre del fichero, como se ve en la figura 26. En futuras actividades del laboratorio necesitará de nuevo excluir ficheros de la compilación, de modo que tenga en mente los pasos que acaba de seguir.

Recuerde además que el fichero de cabeceras `to_7seg.h` también necesita ser modificado cuando la función cuyo prototipo describe está escrita en lenguaje de ensamble pero va a ser llamada desde C++, como es este el caso.

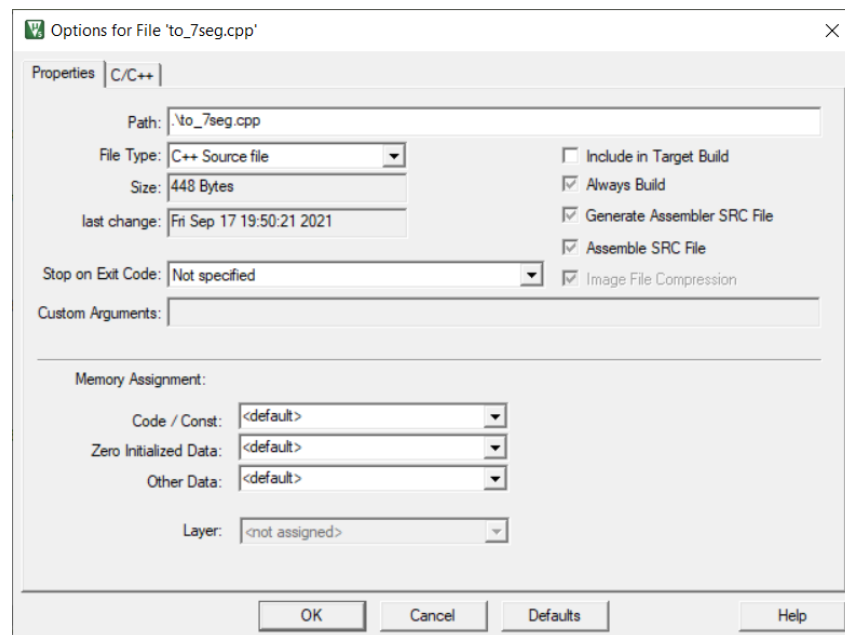


FIGURA 25: exclusión de un fichero de la compilación en *Keil uVision 5*.

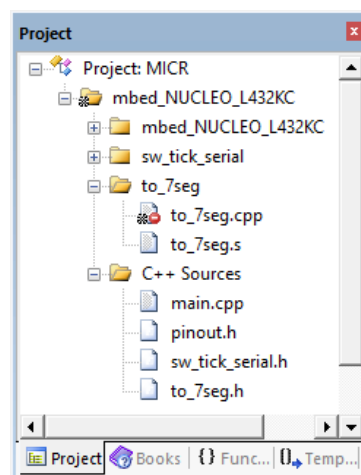


FIGURA 26: indicador de archivo excluido de la compilación en *Keil uVision 5*.

Verifique el funcionamiento del programa corriéndolo sobre la placa y compruebe, haciendo uso del depurador y de la ejecución paso a paso, como la nueva función `to_7seg()` escrita en lenguaje de ensamble es ejecutada.

6.7. Ejercicio 13 - Uso de `printf()` para depuración

Posiblemente haya empleado en asignaturas anteriores la función `printf()` como un recurso para la depuración. Con ella pueden enviarse mensajes de diagnóstico a la consola, lo que suele ser un medio cómodo y rápido de ayuda a la depuración.

Sin embargo en muchos sistemas basados en microprocesador no se dispone de la infraestructura necesaria para poder emplear dicha función, por ejemplo, el sistema puede carecer de pantalla, con lo que no existe medio a través del cual `printf()` pueda mostrar mensajes.

Sin embargo, si el *target* se encuentra conectado a un *host* que sí dispone de pantalla, puede hacerse que `printf()` envíe los mensajes al *host* para que sean mostrados en ella (operación genérica que recibe el nombre de *retargeting* y que también se puede aplicar al teclado). En este laboratorio puede hacerse que un `printf()` que se ejecute en la placa STM envíe, a través del cable USB, los mensajes al PC, donde pueden ser visualizados mediante un programa adecuado. De este modo puede emplearse `printf()` para mostrar mensajes de depuración, mensajes que serán mostrados en el PC.

Para poder usar `printf()` para este cometido debe llamarse a la función `sw_tick_serial_init()` de la librería `sw_tick_serial`, como se ha venido haciendo hasta ahora. A partir de esa llamada, la función `printf()` redirigirá los mensajes a través del puerto USB hasta el PC. Más adelante aprenderá a usar este mecanismo de comunicación sin necesidad de la librería `sw_tick_serial`.

Para poder visualizar en el PC los mensajes recibidos a través del puerto serie se emplean programas denominados *terminales*. Existen una multitud de ellos, siendo el elegido en este laboratorio el programa, gratuito, *Tera Term*. Se describe ahora brevemente la configuración y funcionamiento básicos de esta aplicación.

Una vez arrancado dicho programa éste muestra el diálogo de nueva conexión. En él debe seleccionarse la conexión serie* (Serial) y elegir el puerto del PC a través del cual se realiza la conexión con la tarjeta (que

* El concepto de conexión serie será explicado en las clases de teoría más adelante. Limítese ahora a configurar el programa terminal tal y como se describe. Más adelante entenderá el significado los parámetros que definen esta conexión.

para las placas STM aparecerá como STMicroelectronics STLink Virtual COM Port), tal como se ve en la figura 27, validando con OK.

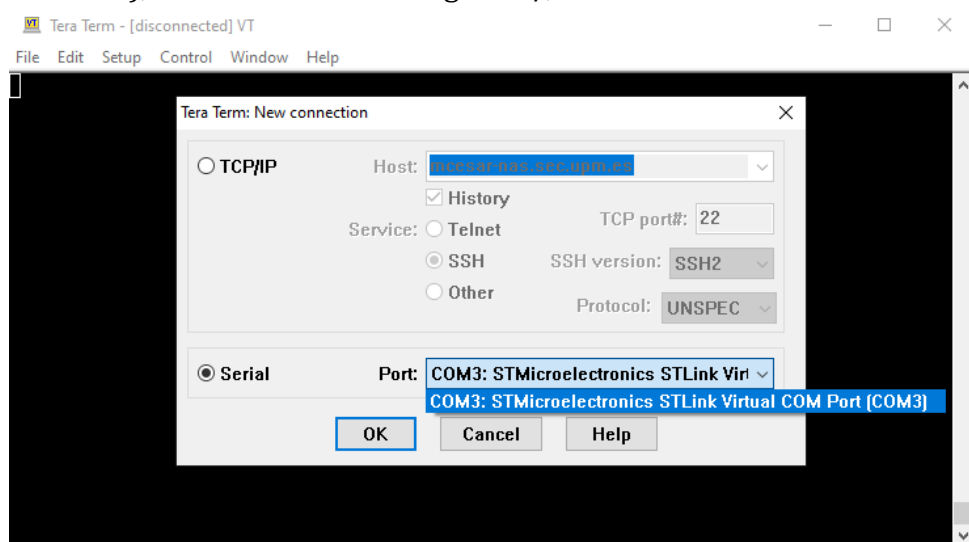


FIGURA 27: diálogo de nueva conexión de *Tera Term*.

Una vez hecho esto es necesario configurar unos parámetros de la conexión empleada. Para ello, dentro del menú Setup → Serial Port... se establecerán dichos parámetros (son: Baud rate, Data, Parity, Stop y Flow control), tal como se ve en la figura 28, validando con OK.

Es necesario también configurar la forma en que el programa terminal procesará los indicadores de fin de línea enviados por la aplicación. Mientras que usualmente se termina cada línea de un mensaje de `printf()` con un carácter de *cambio de línea* —line feed, "\n"—, la interfaz de *Windows* espera que cada línea se termine con una combinación de dos caracteres de control: *retorno de carro* (carriage return) más *cambio de línea* —"\r\n" en C/C++—. Los sistemas operativos basados en *Unix* (esto incluye las versiones de *Mac OS* posteriores a la 9) emplean el primer criterio (sólo "\n"), mientras que *Windows* emplea el segundo ("\r\n"). Con el fin de que *Tera Term* muestre adecuadamente (en una plataforma *Windows*) los mensajes terminados con "\n", es necesario que convierta automáticamente los "\n" recibidos en los "\r\n" esperados por *Windows*. Para ello, en el menú Setup → Terminal..., se establecerá como AUTO el procesamiento de los caracteres de cambio de línea recibidos (en New-line → Receive), tal como se ve en la figura 29. Se validará con OK.

PRÁCTICA 1: ENTORNO, ENSAMBLE, I/O BÁSICO Y EVENTOS

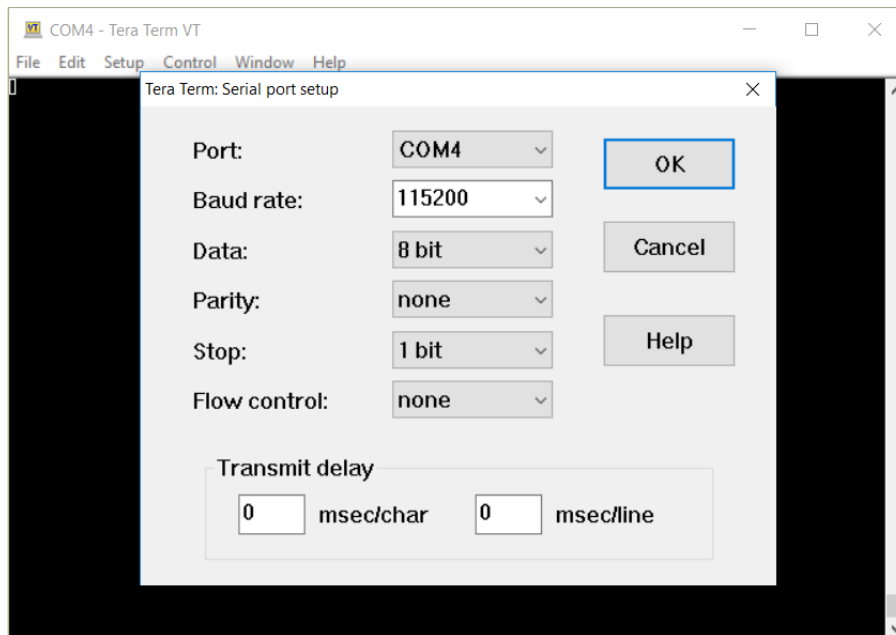


FIGURA 28: configuración de los parámetros de la conexión serie en *Tera Term*.

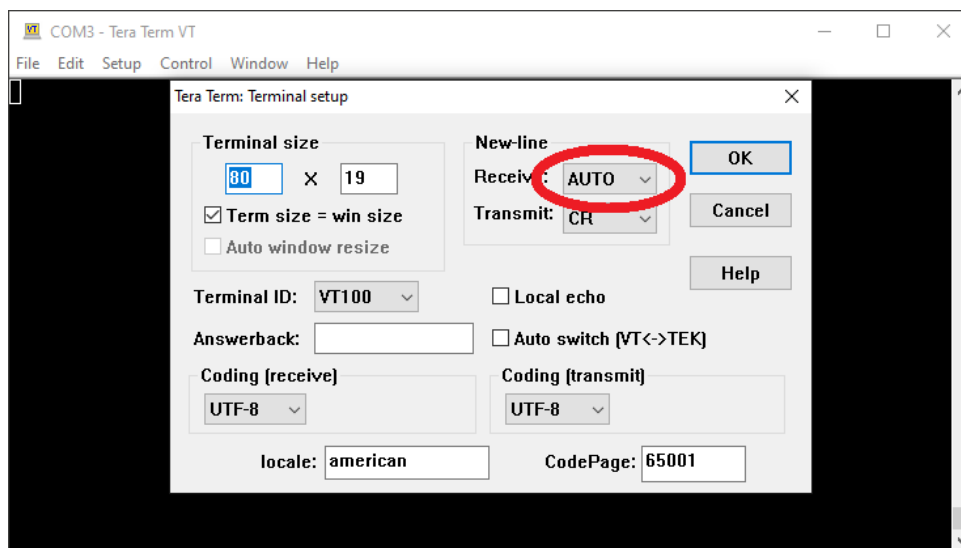


FIGURA 29: configuración para la conversión automática de "\n" a "\r\n" de los caracteres de fin de línea recibidos por *Tera Term*.

Una vez hecho esto, queda *Tera Term* preparado para la recepción y muestra de los mensajes enviados por `printf()` desde la placa STM.

Copie el proyecto de *Keil μ Vision 5* del apartado 6.6 (ejercicio 12) dentro de la carpeta MICR\P1\S2\E13 y trabaje sobre esta copia. Modifique el programa de modo que se envíe un mensaje de depuración mediante `printf()` para que, cada vez que se actúe sobre alguno de los pulsadores, se muestre un mensaje que indique el pulsador concreto que se ha activado y el contenido del *display* de 7 segmentos, tal y como se ve en la figura 30.

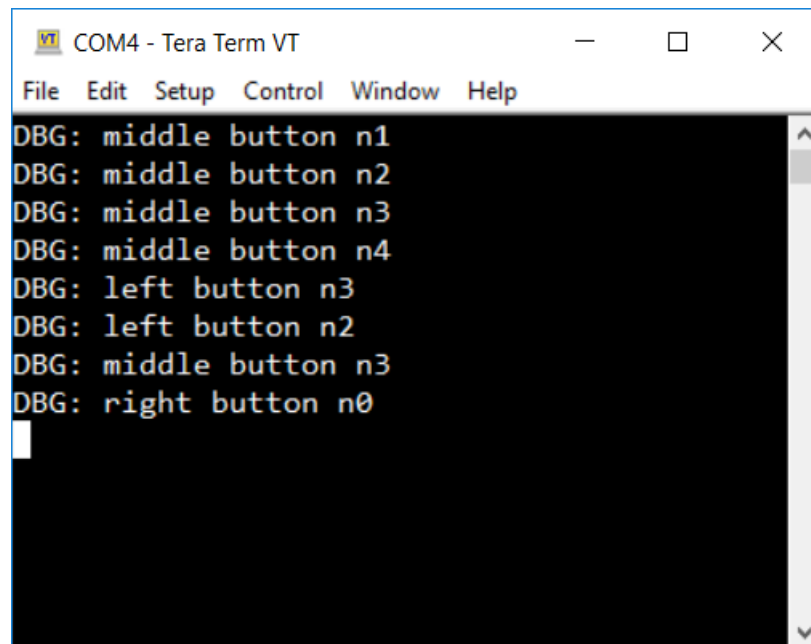


FIGURA 30: mensajes de depuración mostrados por *Tera Term*.

Aunque los mensajes de depuración presentados por `printf()` son útiles durante el desarrollo de la aplicación, no es habitual que tales mensajes sean enviados por la aplicación final una vez terminado el desarrollo. Para evitar la generación de dichos mensajes en la aplicación final sería necesario eliminar todos los `printf()` involucrados antes de la última compilación, lo cual puede ser engorroso. En su lugar se prefiere hacer uso de nuevo de las directivas `#define`, `#ifdef` y `#endif` que se presentaron en la sección 6.4. En particular, cada `printf()` relacionado con la depuración se implementará como:

```
#ifdef VERBOSE
    printf("mensaje_de_depuración\n", ...);
#endif // VERBOSE
```

Si se `#define` antes el símbolo `VERBOSE`, este código se compilará y, durante la ejecución, se mostrarán los mensajes de depuración de `printf()`. En la versión final del programa, en la que no se requieren dichos mensajes, solo es necesario eliminar la línea mediante, por ejemplo:

```
#if 0
# define VERBOSE
#endif
```

y los `printf()` de depuración no serán compilados y, por tanto, no se generarán los mensajes de depuración. Cambiando `#if 0` por `#if 1` se volverán a incluir los mensajes de depuración.

Modifique el programa anterior para que todas las llamadas a `printf()` puedan ser eliminadas por este mecanismo.

Tenga en cuenta que este mecanismo de depuración no sustituye, al más potente y flexible mecanismo de *breakpoints*, ejecución paso a paso y uso de *watches* y *call stack*. Aún más, `printf()` no retorna hasta que ha terminado la transmisión serie del mensaje de depuración, lo que puede requerir un tiempo elevado (digamos del orden de algunos ms, lo que comparado con otros tiempos típicos de la aplicación puede ser un tiempo grande), lo que podría interferir con la ejecución normal de la aplicación.

7. SUBIDA DE RESULTADOS A MOODLE

Elimine todas las carpetas de nombre `~build` y `~listings` de las carpetas de los ejercicios E1 a E13. Elimine igualmente todos los ficheros con extensión `.bin` que se entregaron con la práctica. Comprima entonces la carpeta P1 completa (en formato `.7z`) y súbala a *Moodle* en el enlace correspondiente. Al emplear el programa *7-Zip* para la compresión emplee la opción Añadir al archivo... y, en la ventana que aparece, seleccione en Nivel de Compresión la opción Ultra. Emplee siempre este mecanismo para generar los ficheros comprimidos que subirá a *Moodle*, en caso contrario el fichero comprimido generado podría tener un tamaño excesivo y ser rechazado por *Moodle*.

MICROPROCESADORES

Los resultados de las prácticas no puntúan en la calificación del laboratorio, pero es necesario demostrar mediante estos entregables la realización completa de las prácticas para poder aprobar el laboratorio y la asignatura. Por ello estos entregables son pruebas de evaluación y están sujetos a la normativa vigente en la UPM en cuanto al fraude académico se refiere. No suba entregables que no haya realizado usted personalmente, que no sean originales y tampoco comparta sus resultados con otras personas.

Terminan aquí las tareas a realizar para esta práctica.