

MICROPROCESADORES

PRÁCTICA 2

INTERFACES BÁSICAS DE I/O Y GESTIÓN DE EVENTOS

LED, DISPLAYS DE 7 SEGMENTOS, PULSADORES Y PRINTF()

TITULACIONES DE GRADO DE LA
ETSI DE TELECOMUNICACIÓN



DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

PRIMAVERA 2020 – 2021

© 2020-21 DTE - UPM

ÍNDICE

1. INTRODUCCIÓN	3
1.1. Materiales necesarios para la realización de las prácticas	4
2. PRIMERA SESIÓN	5
2.1. Trabajos previos a la primera sesión presencial	5
2.1.1. PREPARACIÓN DE LAS PLACAS Y DEL ORDENADOR	5
2.1.2. CÁLCULO Y MONTAJE DE LEDS Y DISPLAYS DE 7 SEGMENTOS	6
2.1.3. VERIFICACIÓN DEL MONTAJE Y CARGA DE SOFTWARE	10
2.2. Trabajos durante la primera sesión presencial	11
2.2.1. CICLO DE TRABAJO CON KEIL μ VISION 5	11
2.2.2. EVENTOS	24
2.2.3. CONTROL DEL <i>DISPLAY</i> DE 7 SEGMENTOS	25
2.2.3.1. <i>Adición de groups, ficheros y librerías a un proyecto</i>	26
2.2.3.2. <i>Requisitos para esta aplicación</i>	29
2.2.4. GESTIÓN DE VARIOS EVENTOS	30
3. SEGUNDA SESIÓN	31
3.1. Trabajos previos a la segunda sesión presencial	31
3.1.1. MONTAJE DE PULSADORES Y SEGUNDO DISPLAY	31
3.1.2. VERIFICACIÓN DEL MONTAJE Y CARGA DE SOFTWARE	33
3.2. Trabajos durante la segunda sesión presencial	34
3.2.1. GESTIÓN DE LOS INTERRUPTORES	34
3.2.2. MULTIPLEXACIÓN «LENTA» DE DISPLAYS	34
3.2.3. TRABAJO CON VARIOS FICHEROS FUENTE	35
4. TERCERA SESIÓN	37
4.1. Trabajos previos a la tercera sesión presencial	37
4.2. Trabajos durante la tercera sesión presencial	37
4.2.1. MULTIPLEXACIÓN DE DISPLAYS	37
4.2.2. CREACIÓN DE LIBRERÍAS	40
4.2.3. USO DE <code>PRINTF()</code> PARA DEPURACIÓN	45
4.2.4. SUBIDA DE RESULTADOS A MOODLE	50

1. INTRODUCCIÓN

En este enunciado se describen los trabajos a realizar durante esta segunda práctica. La práctica se divide en tres sesiones, detallándose en las siguientes secciones el trabajo que debe realizarse en cada una de ellas. Adicionalmente, se describen también los trabajos *previos* que deben realizarse con anterioridad a las dos primeras sesiones presenciales de la práctica.

El objetivo de esta práctica es comenzar a emplear la tarjeta de laboratorio y dotarla de una sencilla interfaz de entrada/salida (I/O) basada en unos LED, unos pulsadores y un *display* de 7 segmentos de dos dígitos.

Adicionalmente, se mostrará cómo desarrollar proyectos que empleen diferentes ficheros fuente, cómo incluir librerías de terceros en un proyecto y cómo generar librerías que podrán ser usadas en futuras prácticas. La generación de estas librerías para su uso posterior le ayudará en la futura integración de aplicaciones complejas. También se describirá cómo puede emplearse `printf()` como ayuda a la depuración en un sistema, como son los que va a desarrollar en esta asignatura sobre las placas NXP o STM, que carece de sistema operativo y pantalla.

La técnica empleada en esta y siguientes prácticas para el desarrollo de los programas se basa en la *gestión de eventos*. En esta práctica se hará uso de eventos *independientes* para realizar algunas acciones de control muy sencillas, obviando, de momento, los mecanismos para la generación de dichos eventos. En la práctica 3 se abordará, precisamente, la generación de los eventos mediante el mecanismo de *interrupciones* y en la práctica 4 se contemplará la gestión de *eventos interdependientes* mediante el uso de *máquinas de estados finitos (autómatas) controladas por eventos*.

Adjunto al enunciado encontrará en Moodle un fichero comprimido que, al descomprimirlo (dentro de la carpeta MICR referida en la práctica 1), genera una estructura de carpetas en la que debe ir trabajando y guardando todos los resultados de la práctica. En diferentes partes del enunciado de la práctica se hace mención a carpetas y ficheros incluidos en esta estructura de directorios.

Antes del fin de cada sesión de la práctica deberá comprimir las carpetas MICR\P2 y MICR\to_7seg (que incluyen los resultados de su

trabajo) y subirla a *Moodle*. Elimine antes todas las carpetas ~build y ~listings de todos los proyectos de *Keil μ Vision 5* y también todos los ejecutables de ejemplo (.bin) que se han suministrado. Estos entregables deberán ser subidos a *Moodle* por todos los integrantes de cada puesto de laboratorio. Si no se subiese el entregable o se hiciese después del final de la correspondiente sesión de laboratorio, se entenderá la entrega como no realizada y se calificará con 0 puntos.

1.1. Materiales necesarios para la realización de las prácticas

Con objeto de adelantar la adquisición del material se relacionan a continuación los dispositivos (aparte de la placa NXP o 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 o 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 segunda práctica no se emplearán la LDR ni el sensor telemétrico.

2. PRIMERA SESIÓN

2.1. Trabajos previos a la primera sesión presencial

2.1.1. PREPARACIÓN DE LAS PLACAS Y DEL ORDENADOR

El entorno de desarrollo ya se habrá instalado en la práctica anterior, sin embargo es necesario instalar un *driver* en el ordenador que permite al entorno de *Keil* comunicarse con las tarjetas. 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) mediante:

- el ejecutable `mbedWinSerial_16466.exe` para la placa NXP;
- el *script* `stlink_winusb_install.bat` para las placas STM (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, aunque en este caso los mecanismos de actualización son diferentes:

- para la placa NXP, una vez instalado su *driver*, conectada al ordenador y visible como una unidad de almacenamiento, se copiará en ella el fichero `mbedmicroncontroller_141212.if` y, una vez terminada la copia, se desconectará la placa del puerto USB y se volverá a conectar;
- para la placa STM, una vez conectada 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.

MICROPROCESADORES

2.1.2. 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:

- para el microcontrolador NXP: (incluye además masa, GND):

VOUT	VU	IF-	IF+	RD-	RD+	TD-	TD+	D-	D+	p30	p29	p28	p27	p26	p25	p24	p23	p22	p21
------	----	-----	-----	-----	-----	-----	-----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



GND	VIN	VB	nR	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
-----	-----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

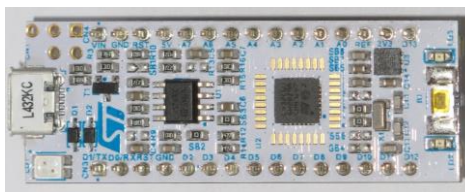
GND

LDL LDM LDR

- y para el microcontrolador STM: (incluye además masa, GND):

GND

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



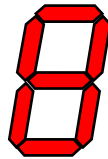
D1	D0	NRST	GND	D2	D3	D4	D5	D6			D9	D10	D11	D12
----	----	------	-----	----	----	----	----	----	--	--	----	-----	-----	-----

LDL LDM LDR

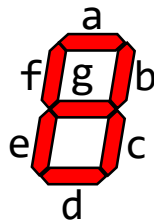
PRÁCTICA 2: INTERFACES BÁSICAS DE I/O

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* de los microcontroladores (disponibles 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 1 se puede ver la representación de varios símbolos alfanuméricos en un *display* de este tipo.

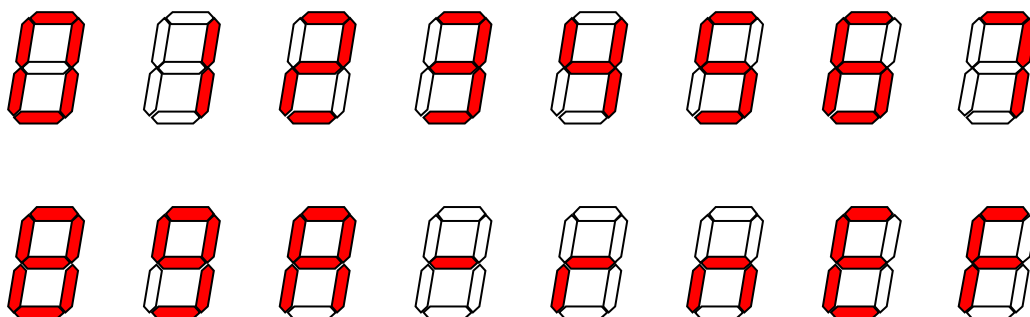


FIGURA 1: representación en un *display* de 7 segmentos de los símbolos {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, -, r, n, E, F}.

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:

- 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 antes del inicio de la primera sesión de esta práctica 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*);

PRÁCTICA 2: INTERFACES BÁSICAS DE I/O

- 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:

- para el microcontrolador NXP:

										SGF	SGG	SGA	SGB	SGE				SGD	
VOUT	VU	IF-	IF+	RD-	RD+	TD-	TD+	D-	D+	p30	p29	p28	p27	p26	p25	p24	p23	p22	p21

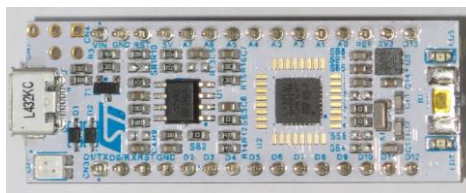


GND	VIN	VB	nR	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
-----	-----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

SGC

- y para el microcontrolador STM:

				SGF SGG SGA SGB				SGE				SGD		
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
----	----	------	-----	----	----	----	----	----	--	--	----	-----	-----	-----

SGC

2.1.3. 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. Borre cualquier fichero .bin que pudiera haber en la unidad, desconecte la alimentación y vuelva a conectarla. Después copie el fichero ejecutable .bin adjunto a la práctica (carpeta Previo_S1, emplee el correspondiente a su placa) en la unidad, desconecte el cable USB y vuelva a conectarlo. 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 1* de la página 8. A la vez, los LED mostrarán, a la misma cadencia, la cuenta binaria ascendente de 3 bits.

Al inicio de la primera sesión de la práctica cada puesto de laboratorio deberá entregar, en papel, el cálculo de las resistencias de limitación de los LED (R_{LED}) y el cálculo de las resistencias de limitación de los segmentos (R_{SEG}), haciendo constar: los valores de todos los parámetros relevantes en el cálculo; las fuentes de las que se han tomado estos (página y tabla de qué *datasheet*, incluya enlaces a las *datasheet* de LED y *displays* que haya empleado); y la forma en que se realiza el cálculo. El docente también comprobará, al inicio de la sesión, que el montaje de todo lo anterior se ha realizado y que la aplicación de prueba (que ya debe venir cargada en la placa) funciona correctamente.

* Este programa emplea, para representar el símbolo «1», una configuración de segmentos diferente a la dada en la figura 1 de la página 7 (enciende los 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 1.

2.2. Trabajos durante la primera sesión presencial

2.2.1. CICLO DE TRABAJO CON KEIL μ VISION 5

En este apartado se pretende emplear una aplicación previamente desarrollada para conocer la metodología de trabajo con el entorno de desarrollo de *Keil*. Para ello se debe abrir el proyecto de ejemplo que se encuentra en *Moodle* (MICR\P2\S1\1_Ejemplo_1) haciendo doble *click* sobre el *fichero de proyecto*, cuya extensión es *.uvprojx*. Se abrirá un entorno de desarrollo como el mostrado en la figura 2. 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.

Como recordará de la práctica anterior y de las clases de teoría, en el caso de sistemas empujados 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 su placa *NXP mbed LPC1768* o *STM Nucleo-l432kc*. Es posible configurar un proyecto de *Keil μ Vision 5* de modo que pueda generar una misma aplicación para distintos *targets*. El proyecto de ejemplo que se le ha entregado trabaja de esta forma y es capaz de generar la aplicación indistintamente para cualquiera de las dos placas antedichas. Pero siempre es necesario indicar a *Keil μ Vision 5* sobre qué *target* se desea trabajar. En la figura 3 se muestran los controles pertinentes.

MICROPROCESADORES

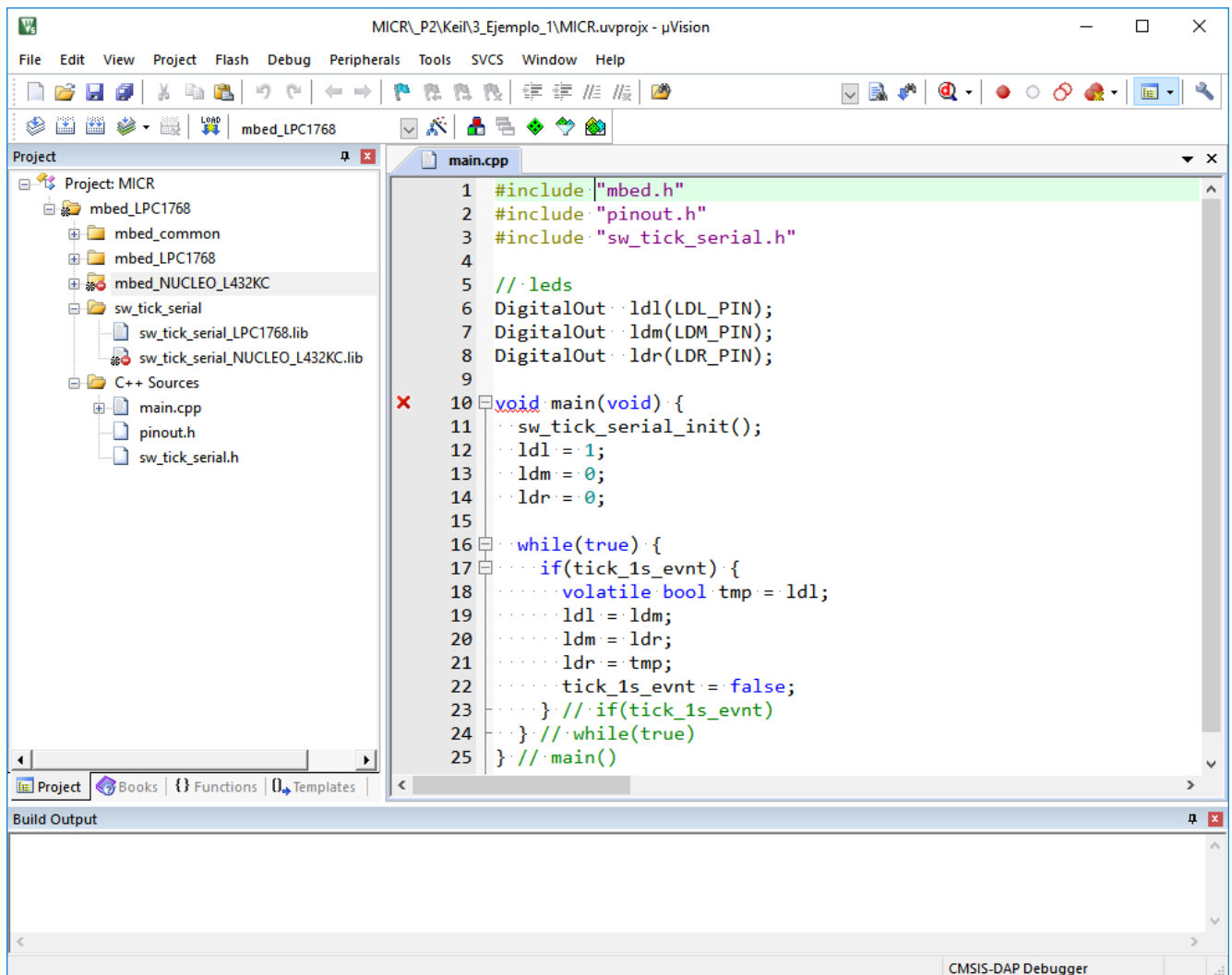


FIGURA 2: entorno de desarrollo Keil μ Vision 5.

- El selector de *target* permite elegir sobre qué *target* se desea trabajar. Las opciones configuradas en los proyectos de esta asignatura son:
 - mbed_LPC1768. Para generar aplicaciones y depurar sobre las placas *NXP mbed LPC1768*, empleando además la librería *mbed* que se describe en las clases de teoría.
 - mbed_NUCLEO_L432KC. Para generar aplicaciones y depurar sobre las placas *STM Nucleo-l432kc*, empleando, como antes, la librería *mbed* que se describe en las clases de teoría.

PRÁCTICA 2: INTERFACES BÁSICAS DE I/O

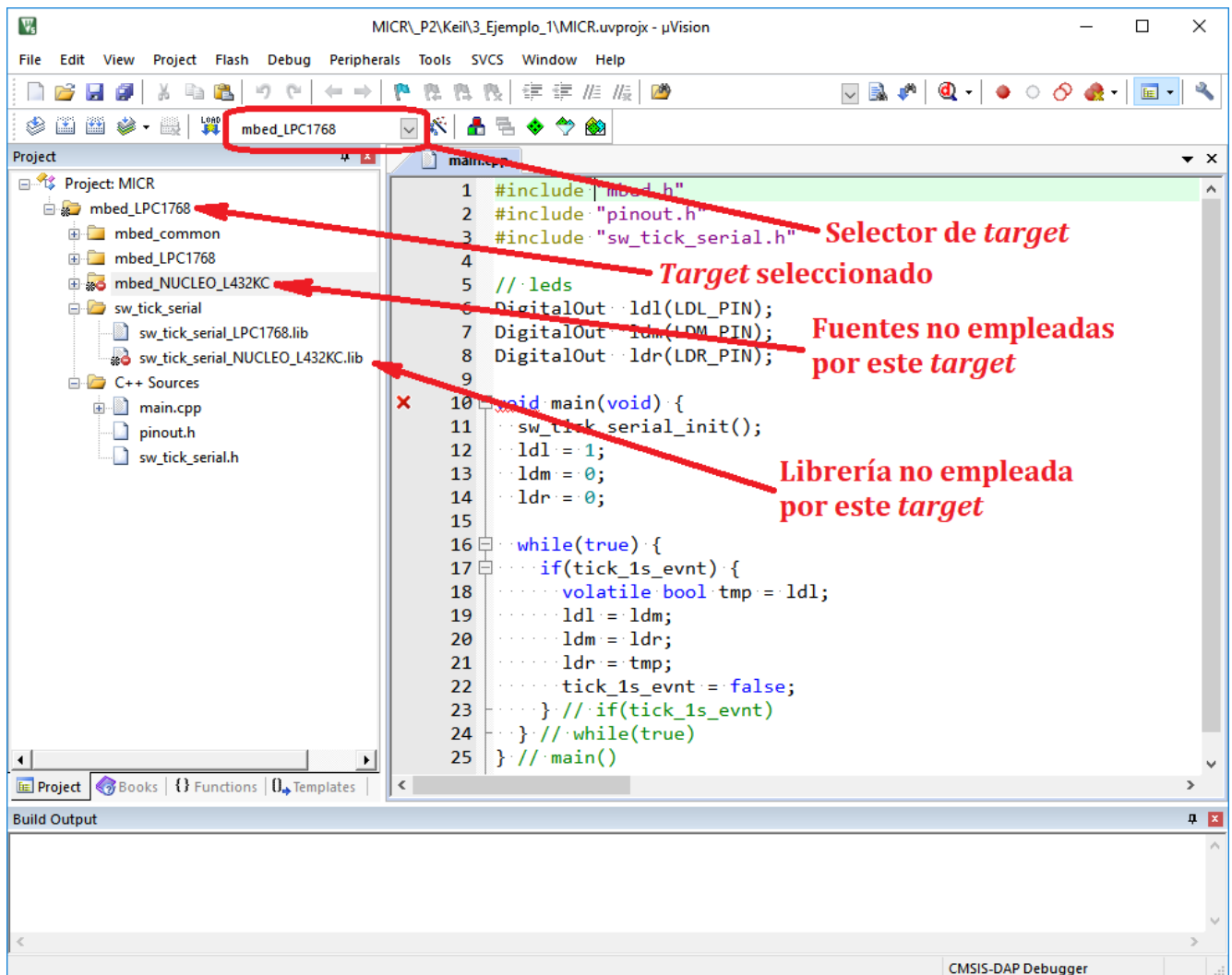




FIGURA 3: selección de target en Keil μ Vision 5.


Debe siempre ajustar el selector de *target* de acuerdo a la placa que esté empleando en el laboratorio.

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_common. Parte de la librería *mbed* común a todos los *targets* soportados por esta librería.




- `mbed_LPC1768`. Parte de la librería *mbed* requerida únicamente por las placas *NXP mbed LPC1768*.
- `mbed_NUCLEO_L432KC`. Parte de la librería *mbed* requerida únicamente por las placas *STM Nucleo-l432kc*.
- `sw_tick_serial`. Librería *sw_tick_serial*, de uso exclusivo durante esta práctica (y en el primer examen de laboratorio).
- C++ Sources. Fuentes en C++ de la aplicación, comunes a todos los *targets*.

Observe que al seleccionar un determinado *target* algunos *groups* se marcan con un pequeño icono de dirección prohibida  que indica que ese *group* no será tenido en cuenta durante la compilación. Lo mismo ocurre con algunos ficheros dentro de un *group*, como se puede apreciar en la figura 3. Pruebe a modificar el selector de *target* para ver como los iconos  se modifican de acuerdo a las necesidades de cada *target*. Esto permite ajustar los ficheros fuentes necesarios para cada *target*, puesto que *Keil μVision 5* no permite que diferentes *targets* tengan diferentes *groups* o ficheros fuente, los *groups* y ficheros son comunes a todos los *targets* y solo se permite excluirlos de la compilación mediante este método. Para excluir o no un fichero o *group* de la compilación de un determinado *target*, seleccione ese *target* y pulse sobre el *group* o fichero con el botón derecho del ratón, selecciona ahora Options for... y, en la pestaña Properties, seleccione o no la opción Include in Target Build.


Al seleccionar diferentes *targets* también se modifican algunas otras opciones del proyecto, en general todas las opciones accesibles desde el menú Options for Target...  son propias del *target* seleccionado y no afectan a otros *targets*. Por ejemplo, el depurador empleado para cada *target* (que aparece en la esquina inferior de la ventana de *Keil μVision 5*, *CMSIS-DAP Debugger* en la figura 3) se ajusta de este modo y es propio para cada *target*.

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 2.1.2 respecto a los pines de la placa *NXP* o *STM* a los que deben conectarse los LED (tendrá que consultar también el fichero

pinout.h y tener en cuenta además que al seleccionar el *target* mbed_LPC1768 se define el macro TARGET_MBED_LPC1768, mientras que si se selecciona el *target* mbed_NUCLEO_L432KC el macro que se define es TARGET_NUCLEO_L432KC —esto puede comprobarse y configurarse dentro del menú Options for Target...  → C/C++ → Define—. Más adelante se describirá con detalle el mecanismo de inclusión condicional de código mediante las directivas del preprocesador `#define`, `#ifdef`, `#ifndef` y `#endif` que se emplea en 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 en la práctica anterior. 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 la práctica anterior 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 pueden 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 la práctica anterior, 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 4 para el *target* mbed_LPC1768 y en la figura 5 para el *target* mbed_NUCLEO_L432KC. Observe en estas figuras cómo la opción Use Simulator aparece desmarcada y, en su lugar, aparecen marcadas las opciones:

- Use: CMSIS-DAP Debugger para el *target* mbed_LPC1768, que es el tipo de depurador incluido en la placa NXP *mbed* LPC1768;
- Use: ST-Link Debugger para el *target* mbed_NUCLEO_L432KC., que es el tipo de *debugger* incluido en las placas STM Nucleo.

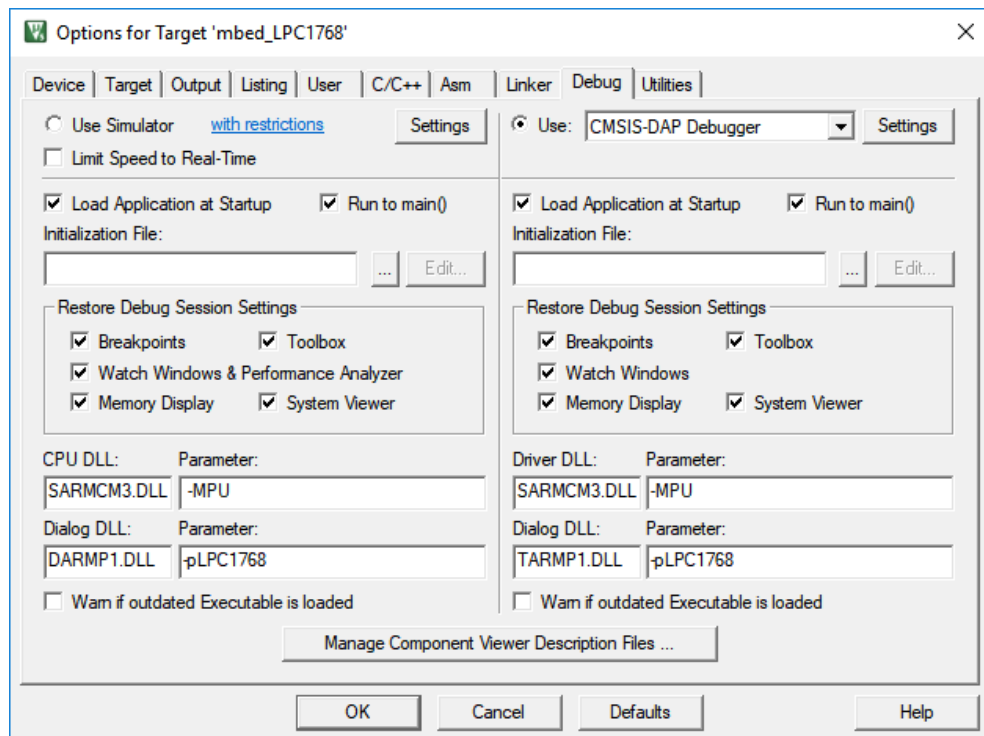


FIGURA 4: configuración de depuración para el *target* mbed_LPC1768.

Asegúrese de que todas las opciones en la parte derecha de esta pestaña se corresponden con las vistas en las figuras 4 y 5.

También deben ajustarse algunos parámetros del *debugger* accesibles desde Settings → Debug de la pestaña Debug. Dichos parámetros se encuentran descritos en las figuras 6 y 7, destacados en rojo. Estos ajustes básicamente determinan qué interfaz concreta de depuración emplear (en ambos casos la interfaz SWD en vez de la JTAG), su frecuencia máxima de funcionamiento (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 las figuras 6 y 7.

PRÁCTICA 2: INTERFACES BÁSICAS DE I/O

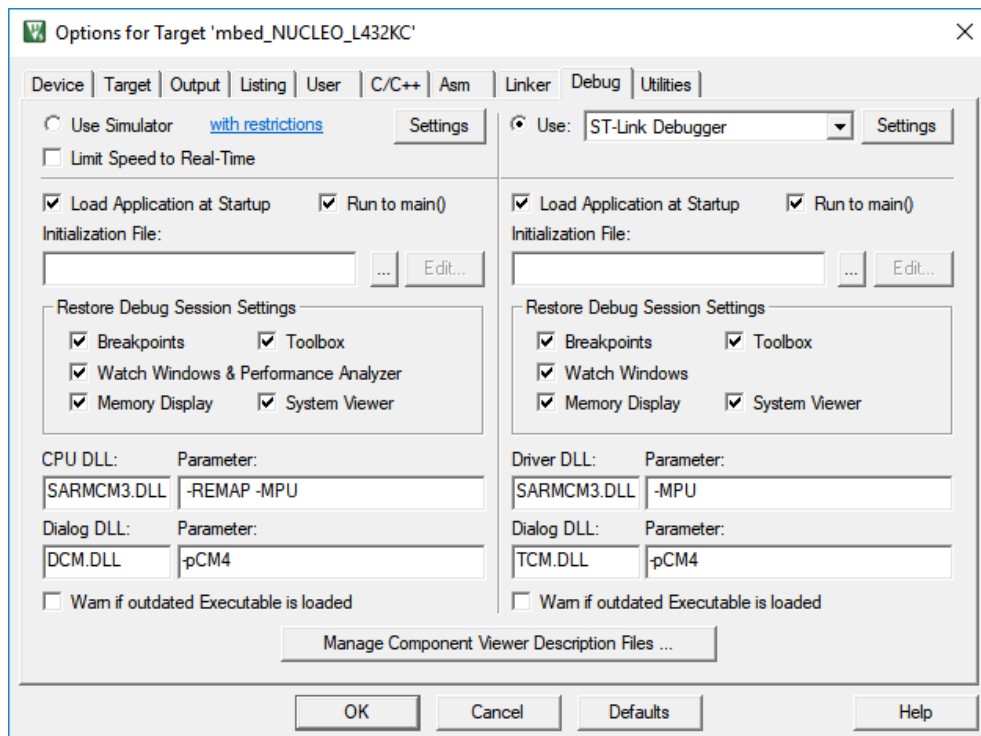


FIGURA 5: configuración de depuración para el target mbed_NUCLEO_L432KC.

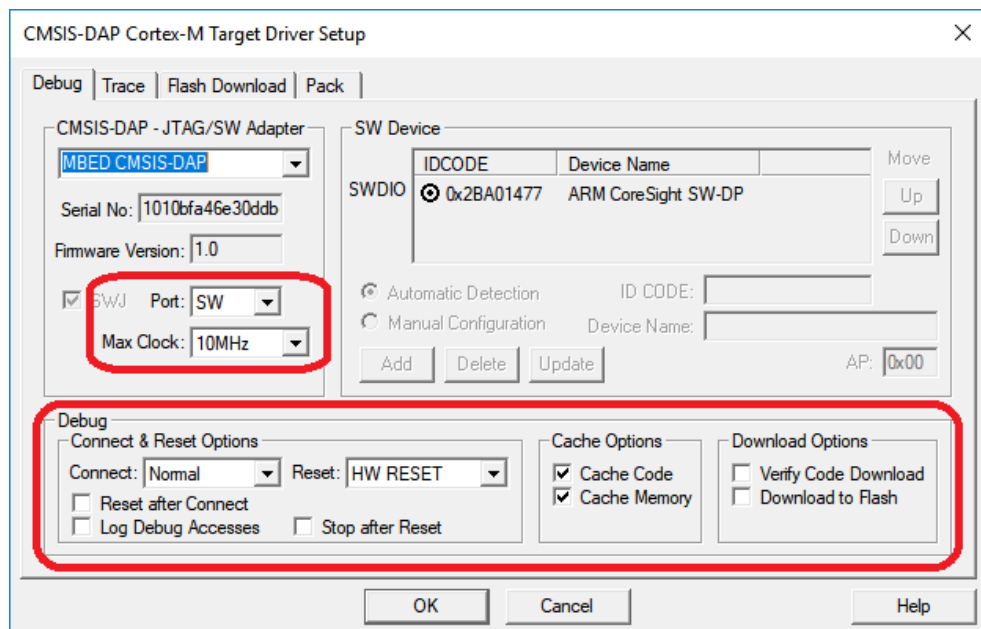


FIGURA 6: ajustes del debugger para el target mbed_LPC1768.

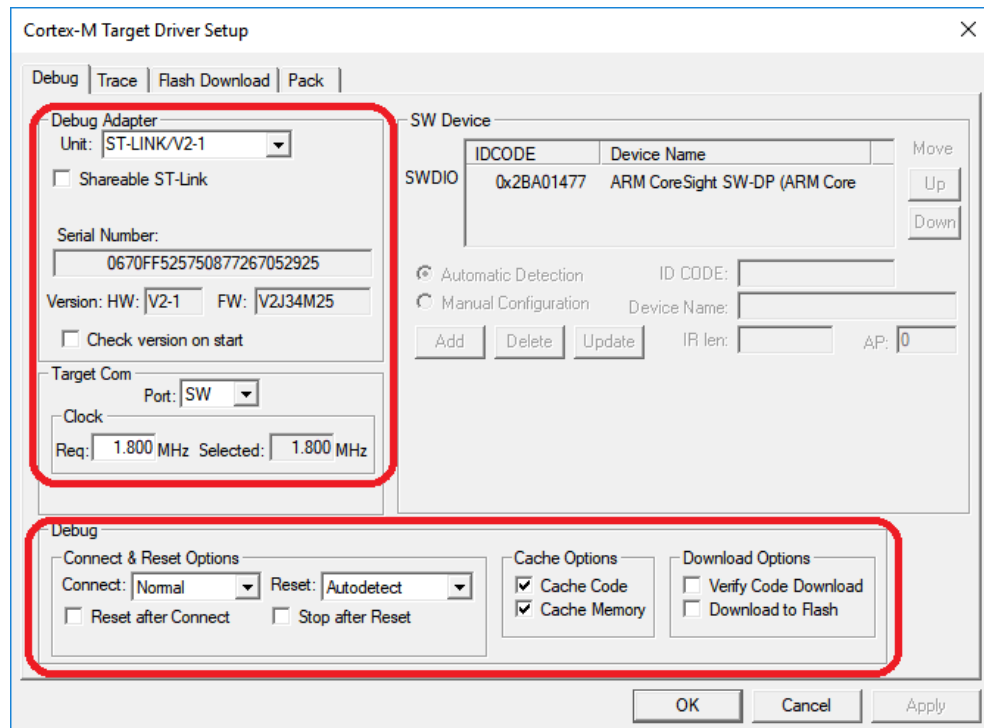



FIGURA 7: ajustes del *debugger* para el *target* mbed_NUCLEO_L432KC.

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 de Options for Target , asegúrese de que está seleccionado Use Target Driver for Flash Programming y activas las opciones Use Debug Driver y Update Target before Debugging, como ve en la figura 8. 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. Estos algoritmos son:

- LPC17xx IAP 512kB *Flash* para el *target* mbed_LPC1768.
- STM32L4xx 256 KB *Flash* para el caso del *target* mbed_NUCLEO_L432KC.

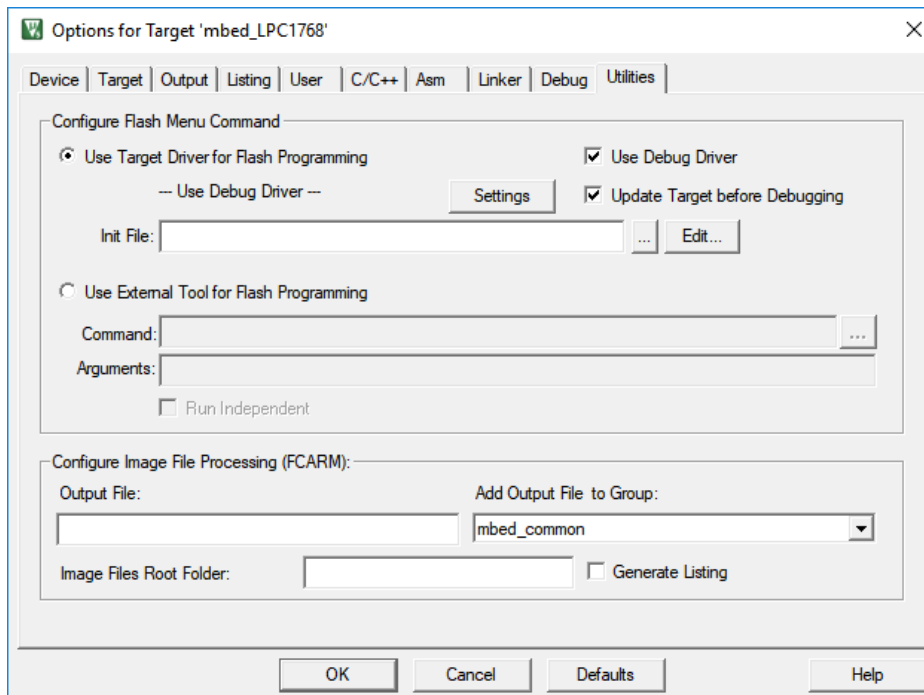


FIGURA 8: configuración del mecanismo de volcado a la memoria *Flash* en *Keil uVision 5*.

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 las figuras 9 y 10 se muestran estos ajustes para ambos *targets*. Revise estos ajustes si la herramienta *Keil uVision 5* le informara de errores al volcar el programa en las placas.

MICROPROCESADORES

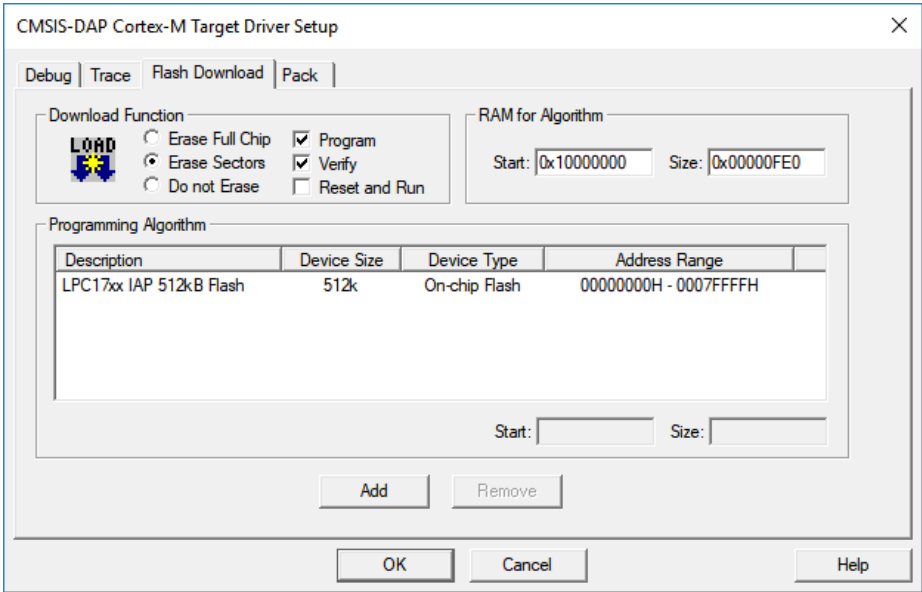


FIGURA 9: ajustes del método de programación de la memoria *Flash* para el *target* mbed_LPC1768.

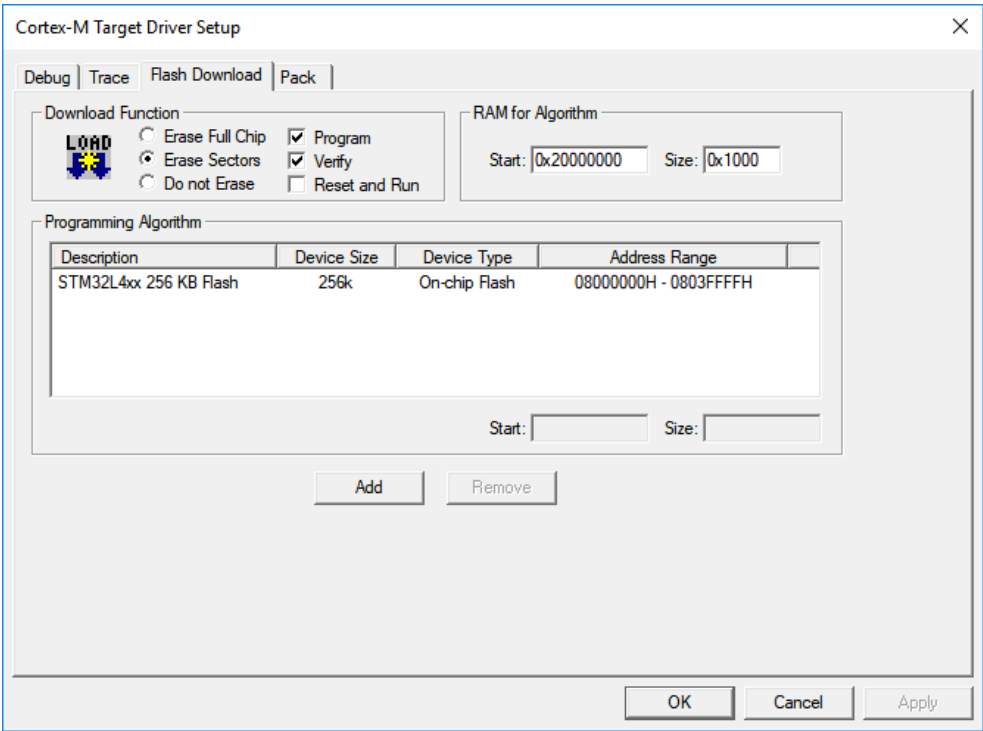















FIGURA 10: ajustes del método de programación de la memoria *Flash* para el caso del *target* mbed_NUCLEO_L432KC.

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 la práctica anterior):

- 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 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.

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 11 se muestra como añadir un Watch para la variable `tmp`.

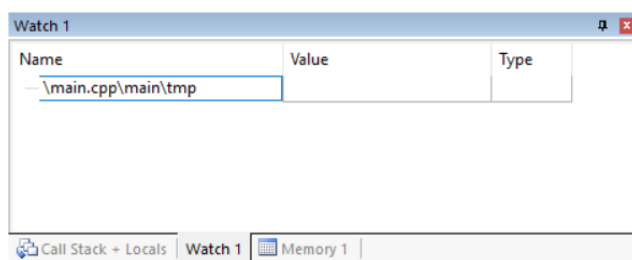






FIGURA 11: añadiendo una variable, mediante su nombre completo, a una ventana de Watch en Keil μ Vision 5.

Para este primer ejemplo:

1. compile el programa;
2. pase a modo de depuración;
3. lance la ejecución continua (botón ) , observe los LED;
4. ponga un punto de ruptura en la línea 20 (`g_ldl = 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 (botón ) —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 determinara, 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. El calificativo **volatile** tiene otros usos que explorará más adelante. El mecanismo de *watch* también podría no mostrar variables que, en el punto donde se ha interrumpido la ejecución, no se encuentren dentro del *alcance* (*scope*) del programa.

Si no consigue realizar una ejecución paso a paso o modificar el valor de la variable indicada en el punto 7 es conveniente que consulte al docente.

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 = 4;
    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. 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.

2.2.2. 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 3 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.

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.


2.2.3. 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\P2\S1\2_Cuenta 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 2.1.3.

En la carpeta MICR\P2\S1\2_Cuenta encontrará también 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 NXP o STM.

2.2.3.1. 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, en realidad hay dos, uno para cada *target*). 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\P2\S1\2_Cuenta). 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 12.
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 13. 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. El fichero se añade al *group* para todos los *targets*, no es necesario repetir estas operaciones para cada *target*.

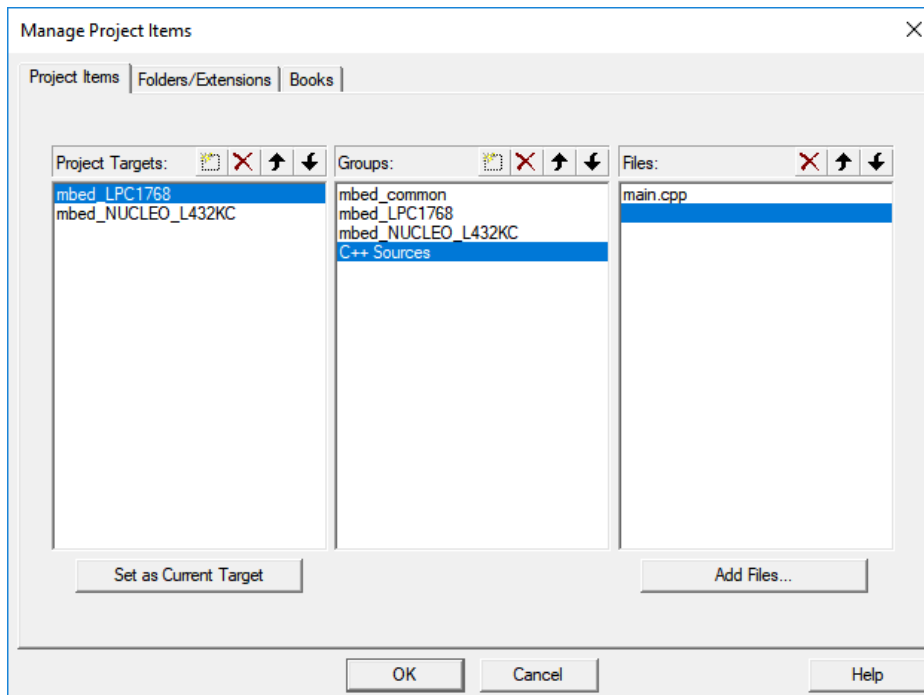


FIGURA 12: pestaña Project Items.

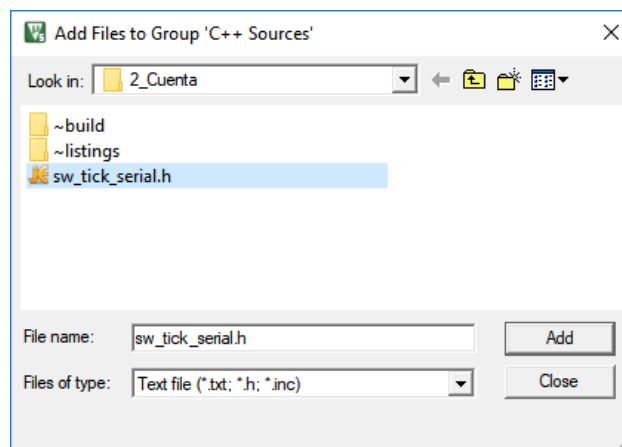





FIGURA 13: diálogo Add Files to Group.

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 de los ficheros `.lib` (objeto), que también deben ser incluidos en el proyecto. En este caso se va a crear un *group* específico para ellos 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.

5. Tal como hizo en el punto 3 anterior, añada ahora los dos ficheros .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 los ficheros objeto de la librería sw_tick_serial al proyecto dentro de un *group* llamado sw_tick_serial, como se aprecia en la figura 14.
6. Se acaban de añadir dos ficheros .lib al proyecto, uno para cada *target*. Es necesario indicar a Keil μ Vision 5 a qué *target* va asociado cada uno. Para ello seleccione el *target* mbed_LPC1768 y picando con el botón derecho sobre el fichero no asociado a este *target* (es decir, sobre sw_tick_serial_NUCLEO_L432KC.lib) seleccione Options for File.... En la ventana emergente desmarque la opción Include in Target Build. Termine con OK. Notará que aparece el icono  sobre el nombre del fichero, indicando que no será empleado durante la compilación del target mbed_LPC1768, como se ve en la figura 3 de la página 13. Repita estas operaciones para que el *target* mbed_NUCLEO_L432KC excluya el fichero sw_tick_serial_LPC1768.lib.

A partir de la realización de esta tarea, todos los proyectos de Keil μ Vision 5 que entregue deben estar adecuadamente configurados para cada uno de los dos *targets* indicados, en caso contrario no podrán ser corregidos y serán calificados con 0 puntos.

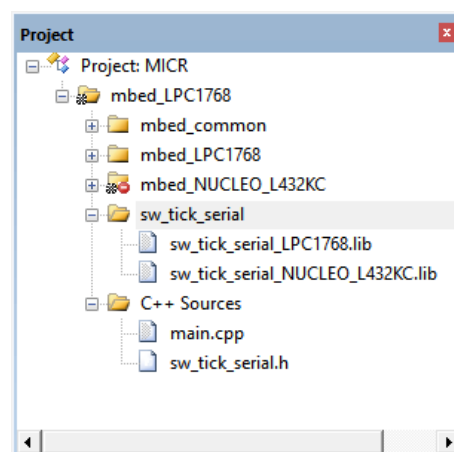


FIGURA 14: *group* y archivos de sw_tick_serial añadidos al proyecto.

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 práctica anterior 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.

2.2.3.2. Requisitos para esta aplicación

Las 7 señales SGA a SGG que controlan los segmentos del *display* se agruparán en un `BusOut`. 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 1 en la página 8). El prototipo de dicha función será:

```
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:

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

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.

Deberá enseñar al docente su programa funcionando.

2.2.4. 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.

Se recomienda que copie proyecto del apartado anterior en `MICR\P2\S1\2_Cuenta` (no incluya los `.bin` que se suministraron) dentro de la carpeta `MICR\P2\S1\3_Cuenta_y_LED` y que realice este apartado en esta carpeta. En `MICR\P2\S1\3_Cuenta_y_LED` encontrará un fichero ejecutable `.bin` que puede emplear para observar el funcionamiento esperado para este apartado.

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

3. SEGUNDA SESIÓN

3.1. Trabajos previos a la segunda sesión presencial

3.1.1. 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:

- para el microcontrolador NXP:

VOUT	VU	IF-	IF+	RD-	RD+	TD-	TD+	D-	D+	p30	p29	p28	p27	p26	p25	p24	p23	p22	p21
------	----	-----	-----	-----	-----	-----	-----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



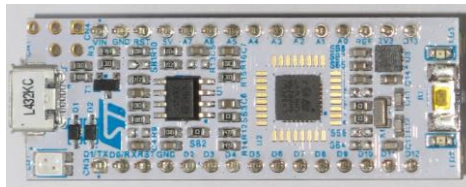
GND	VIN	VB	nR	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
-----	-----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

SWL SWM

SWR

- y para el microcontrolador STM:

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 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 Wilmschurst 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:

- para el microcontrolador NXP:

DSL DSR

VOUT	VU	IF-	IF+	RD-	RD+	TD-	TD+	D-	D+	p30	p29	p28	p27	p26	p25	p24	p23	p22	p21
------	----	-----	-----	-----	-----	-----	-----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



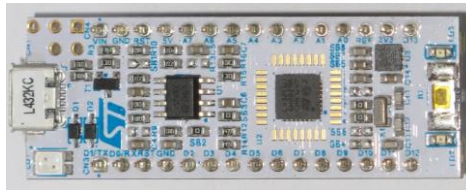
GND	VIN	VB	nR	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
-----	-----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

PRÁCTICA 2: INTERFACES BÁSICAS DE I/O

- y para el microcontrolador STM:

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).

3.1.2. 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\P2\Previo_S2). 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.

Al inicio de la segunda sesión de la práctica el docente comprobará que el montaje de todo lo anterior se ha realizado y que la aplicación de prueba funciona correctamente.

3.2. Trabajos durante la segunda sesión presencial

3.2.1. 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 3.1.1, 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 µVision 5* dentro de la carpeta MICR\P2\S2\4_Switches y trabaje sobre esta copia.

Deberá enseñar al docente su programa funcionando.

3.2.2. 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\P2\S2\5_Lenta 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.

3.2.3. 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\P2\S2\6-Headers 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 —incluida `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 1 de la página 8 (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 2.2.3.1), 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 del laboratorio.

Deberá enseñar al docente su programa funcionando una vez separado en los dos ficheros fuente descritos.

Terminan aquí las tareas a realizar durante la segunda sesión presencial de esta práctica.

4. TERCERA SESIÓN

4.1. Trabajos previos a la tercera sesión presencial

No se planifican actividades previas para esta sesión presencial.

4.2. Trabajos durante la tercera sesión presencial

4.2.1. 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 µVision 5* de las sesiones anteriores (el que considere más útil para este ejercicio) dentro de la carpeta MICR\P2\S3\7_Mux y trabaje sobre esta copia para realizar este programa. En esa misma carpeta encontrará un ejecutable —Mux— que

le permitirá observar sobre la tarjeta el funcionamiento esperado. Encontrará también otro fichero ejecutable —Sombras— 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 3.2.2, 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

* 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.

transistores no puedan conmutar o el procesador no sea capaz de 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 la función `to_7seg()` que creó al final de la sesión anterior, manteniéndola en ficheros (.h y .cpp) separados.

Emplee el osciloscopio, visualizando simultáneamente las señales DSL y DSR, para comprender el motivo de la aparición de las sombras al ejecutar Sombras y la ausencia de las mismas con Mux.

Deberá enseñar al docente su programa funcionando y explicarle, además, la razón por la que el ejecutable Sombras no funciona adecuadamente, justificándolo con medidas realizadas con el osciloscopio.

Para el semestre de primavera del curso 2020-21 la tercera sesión de la práctica 2 no será presencial, por lo que no se requiere que realice las medidas con el osciloscopio ni que explique por qué Sombras no funciona adecuadamente (pero sí deberá enseñar su código). Sin embargo, es conveniente que analice la secuencia de instrucciones que se ejecutan al multiplexar los dos displays para identificar si el orden de tales instrucciones es el adecuado para evitar la aparición de las sombras a las que se refiere el párrafo anterior.

* 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.

4.2.2. CREACIÓN DE LIBRERÍAS

Cuando las aplicaciones incrementan su complejidad, y con el fin de simplificar la labor de creación de la aplicación, suele ser necesario emplear librerías que o bien realiza el propio diseñador o bien son distribuidas por terceros y que «encapsulan» determinadas funcionalidades. Una librería puede estar formada por:

- uno o varios ficheros de cabecera (.h) y uno o varios ficheros fuente (.c/.cpp), tal como vio en el apartado 3.2.3, estas librerías son compiladas con el resto de la aplicación;
- o uno o varios ficheros de cabecera (.h) y uno o varios *ficheros objeto* ya compilados (.o, .a, .lib,...), tal como vio en el apartado 2.2.3.1, estas librerías son *linkadas* (*enlazadas*, *linked*) con el resto de la aplicación;
- o una combinación de ambas cosas.

En el caso de este laboratorio, las librerías *sw_tick_serial* (que solo se usará en esta práctica) y *mbed* (que se usa en esta práctica y en las siguientes) se distribuyen en forma de cabeceras más objetos, la librería *to_7seg* que creó en la sesión anterior, en cambio, es del tipo cabecera más fuentes. En este apartado se pide que convierta la librería *to_7seg* en una del tipo cabecera más objeto y la incluya en un proyecto de *Keil μVision 5*.

Un fichero *objeto* no es más que un conjunto de funciones y variables que han sido compilados para obtener el código binario, pero que no han sido *linkadas* para generar un ejecutable, ya que esa operación de *linkado* (*enlazado*, *link*) se pospone hasta que sean incorporadas a una aplicación que tenga un `main()` y pueda ser ejecutada. En un fichero objeto no hay, por tanto, una función `main()` y en los ficheros fuente que dan lugar a ese fichero objeto tampoco existe un `main()`.

Cuando se emplea una librería lo más habitual es no disponer del código fuente de la misma, para de este modo proteger su creador sus propiedades intelectual e industrial. Por tanto, cuando uno adquiere una librería recibe uno o varios ficheros binarios (objeto) que puede usar desde su aplicación, pero no dispone de los ficheros fuente C/C++ (o cualquier otro lenguaje).

La siguiente cuestión a resolver es ¿cómo se puede, desde una aplicación, utilizar los objetos que esa librería contenga? Para ello debe tenerse acceso a las declaraciones de esos objetos. En el caso de C/C++ esta información se formaliza como prototipos de funciones (que inclu-

yen el nombre de la función y la descripción de sus parámetros y valor devuelto) y como declaraciones de objetos externos —**extern**— (que describen el tipo de cada objeto). Esta información es facilitada (junto con los ficheros objeto) por el desarrollador de la librería mediante un conjunto de ficheros de cabeceras (.h) en el que se declaran todas las funciones y demás objetos que la librería contenga. Adicionalmente, se debe facilitar alguna documentación escrita que indique la funcionalidad de cada función, los parámetros que recibe y devuelve y la utilidad de los restantes objetos contenidos en la librería.

Para acceder desde un programa a los recursos de una librería es necesario *incluir* el fichero (o los ficheros) de cabeceras al comienzo con la directiva del preprocesador **#include** *"nombre_de_la_librería.h"* (esto en C/C++, otros lenguajes presentarán otros mecanismos). A partir de ese momento se puede llamar a las funciones que están incluidas en la librería y referirse a los objetos en ella.

Una vez desarrollado el programa es necesario realizar la compilación del código generado y el *linkado* del mismo con la librería. Este *linkado* puede conseguirse, en el entorno *Keil μVision 5*, añadiendo la librería al proyecto (como se hizo en el apartado 2.2.3.1), al hacer esto dicha librería es automáticamente incluida en el *linkado* de la aplicación. *Keil μVision 5* emplea la extensión .lib para estas librerías.

Es aconsejable que cada librería agrupe todos sus ficheros, dentro del entorno *Keil μVision 5*, en un *group*, como se ve en la figura 15, en la que los archivos objeto de la librería *sw_tick_serial* (un .lib por cada *target*, como sabe) se han incluido dentro del grupo *sw_tick_serial* del proyecto (la librería *mbed* también ocupa sus propios *groups*).

El objetivo de este apartado es que genere una librería del tipo cabeceras más objeto, que se llamará de nuevo *to_7seg*, y que contendrá la función **to_7seg()** que se desarrolló en el apartado 3.2.3. Posteriormente el programa realizado en el apartado anterior será modificado para hacer uso de esta librería, empleando el procedimiento de adición de librerías descrito en 2.2.3.1. Para alcanzar este objetivo deben realizarse los siguientes pasos:

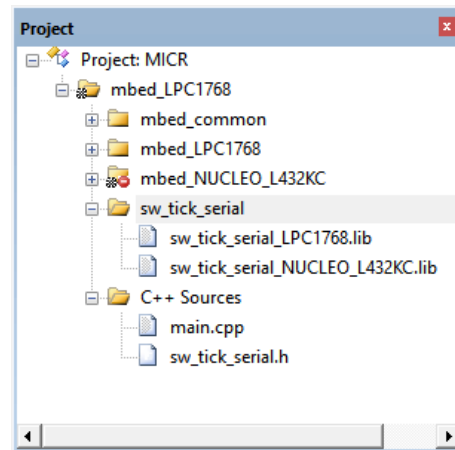



FIGURA 15: librería añadida al proyecto de Keil μ Vision 5.

1. Genere la librería (compilar sin *linkar*) a partir del fichero `to_7seg.cpp` que se creó en el apartado 3.2.3. Para ello debe hacer una copia de aquel proyecto en la carpeta `MICR\P2\S3\8_Lib` y trabajar sobre esta copia. Ese proyecto *so-*lo debe tener un fichero fuente (en su propio *group* C++ Sources), que es el que contiene la función que formará parte de la librería (`to_7seg.cpp`) y deben eliminarse, por tanto, el resto de ficheros, aunque *no* los *groups* de la librería *mbed*, que contienen, entre otras cosas, la definición de los tipos de datos `int8_t` y `uint8_t`, como ya se explicó en la sección 3.2.3. El *group* de la librería `sw_tick_serial` también debe eliminarse. Se deberá también generar el fichero de cabeceras que permita a otros programas emplear las funciones incluidas en la librería. El fichero `to_7seg.h` que creó en el apartado 3.2.3 debe servir, sin modificación, para este fin. Si compila ese proyecto directamente verá que se produce el error:

Error: L6218E: Undefined symbol `$Super$$main`

que indica que no es posible generar un ejecutable ya que no existe una función `main()`. Esto es lógico, ya que el proyecto originalmente se creó para generar un ejecutable y ahora no lo hay. Por tanto, hay que indicar al compilador que lo que se desea es generar una librería, no un ejecutable. Para ello, en Options for Target , pestaña Output, marque la opción Create

Library. Cambie también el nombre del fichero de salida (Name of Executable:) a `to_7seg_LPC1768`. (para el *target* `mbed_LPC1768`) o `to_7seg_NUCLEO_L432KC` (para *target* `mbed_NUCLEO_L432KC`), según se ve en la figura 16.

Será necesario también, en la pestaña User, desmarcar la opción After Build/Rebuild → Run #1 (ver figura 17) que servía para convertir el fichero ejecutable de formato `.axf` (el formato nativo de salida de las herramientas de arm) a formato `.bin`, y que no tiene sentido aquí, al no haber ejecutable.

Estas operaciones, así como las correspondientes compilaciones, deben realizarse para ambos *targets*, de manera que se generen dos ficheros `.lib`, uno para cada *target*.

Si las compilaciones son fructíferas, dentro de la carpeta `~build` deben localizarse los ficheros objeto de esta librería `to_7seg_LPC1768.lib` y `to_7seg_NUCLEO_L432KC.lib`.

2. Cree una carpeta `MICR\to_7seg` y copie en ella los ficheros `.lib` recién obtenidos, así como el fichero de cabecera `to_7seg.h`. Esta será la ubicación de su librería *to_7seg*.
3. Incluya la librería en un proyecto y use su función. Para realizar esta parte del proceso copie el proyecto desarrollado en el apartado 4.2.1 en la carpeta `MICR\P2\S3\9_MuxLib` y trabaje sobre esta copia. Incorpore su librería al proyecto tal como se hizo en el apartado 2.2.3.1 con la librería *sw_tick_serial*. El nuevo *group* para la librería se llamará `to_7seg`, como se ve en la figura 18.
4. No olvide eliminar de este proyecto el fichero fuente `to_7seg.cpp`, que ya no es necesario. Compile la aplicación y verifique el funcionamiento del programa.

Recuerde que el proyecto debe quedar adecuadamente configurado *para los dos targets*, incluyendo la librería *to_7seg* compilada para cada uno de ellos, en caso contrario no podrá ser corregido y será calificado con 0 puntos

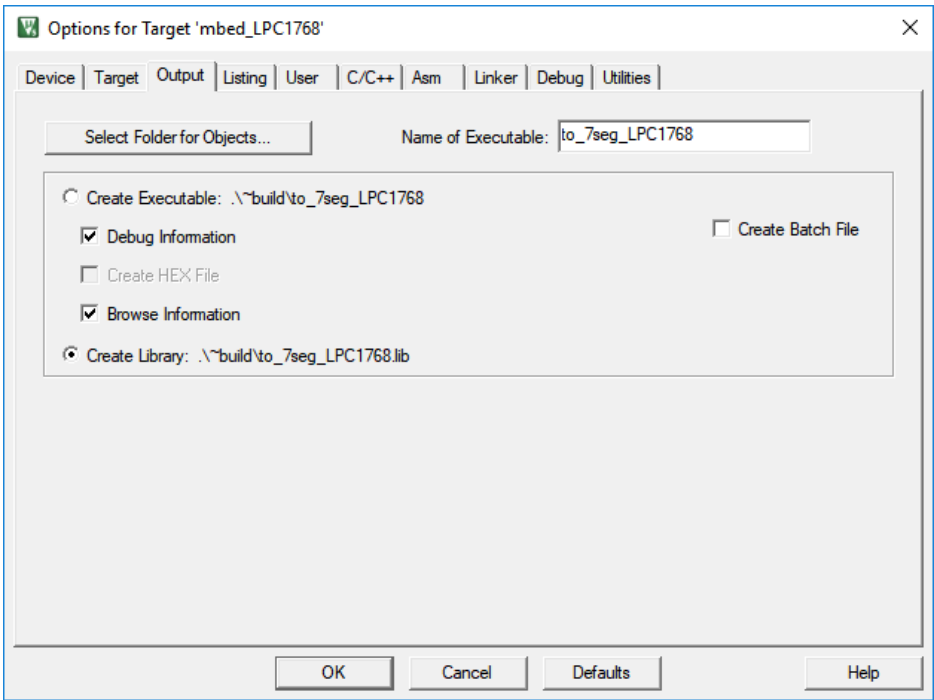


FIGURA 16: generar librería en vez de ejecutable y nombre del fichero de salida.

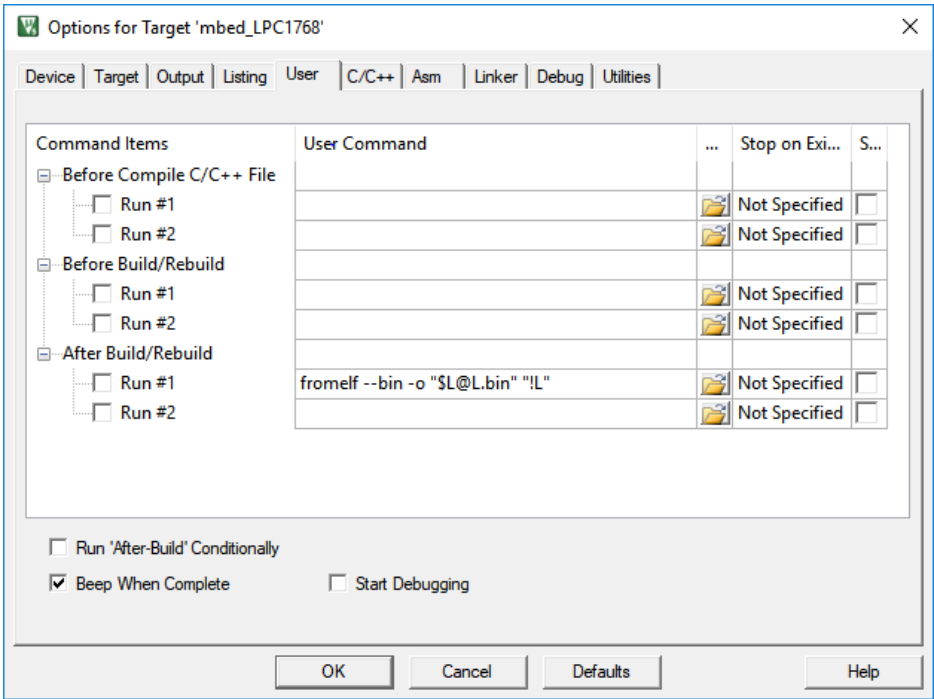


FIGURA 17: inhibir la conversión de formato .axf a .bin.

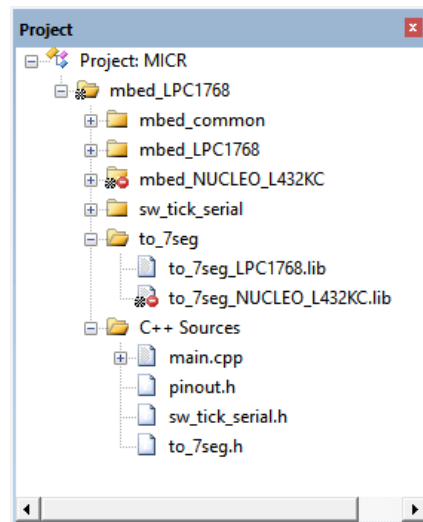


FIGURA 18: adición de la librería to_7seg al proyecto.

4.2.3. 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. En este laboratorio puede hacerse que un `printf()` que se ejecute en la placa NXP o 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 (aparecerá como mbed Serial Port para las placas NXP y como STMicroelectronics STLink Virtual COM Port para las placas STM), tal como se ve en la figura 19, validando con OK.

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 20, 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—. 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 21. Se validará con OK.

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

PRÁCTICA 2: INTERFACES BÁSICAS DE I/O

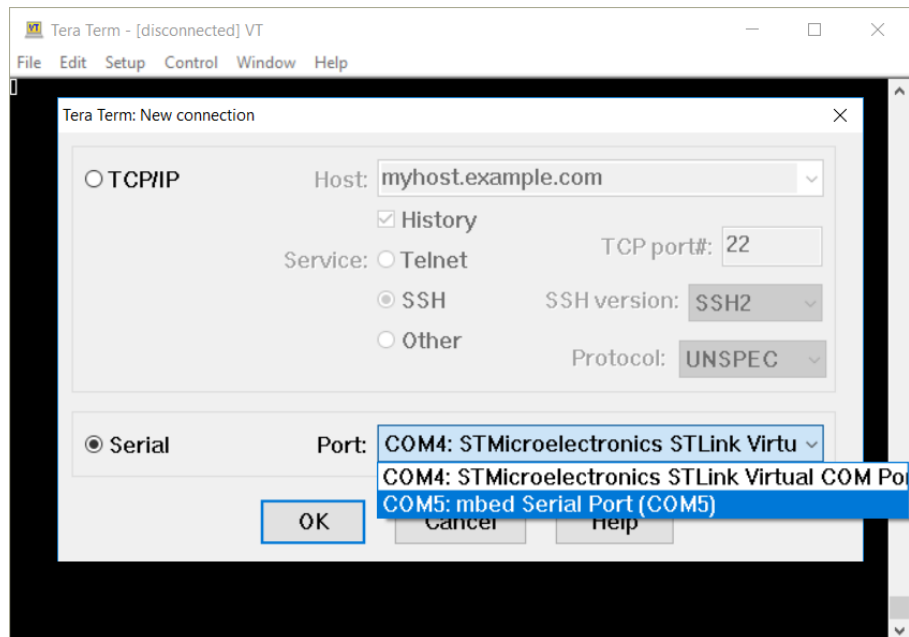


FIGURA 19: diálogo de nueva conexión de *Tera Term*.

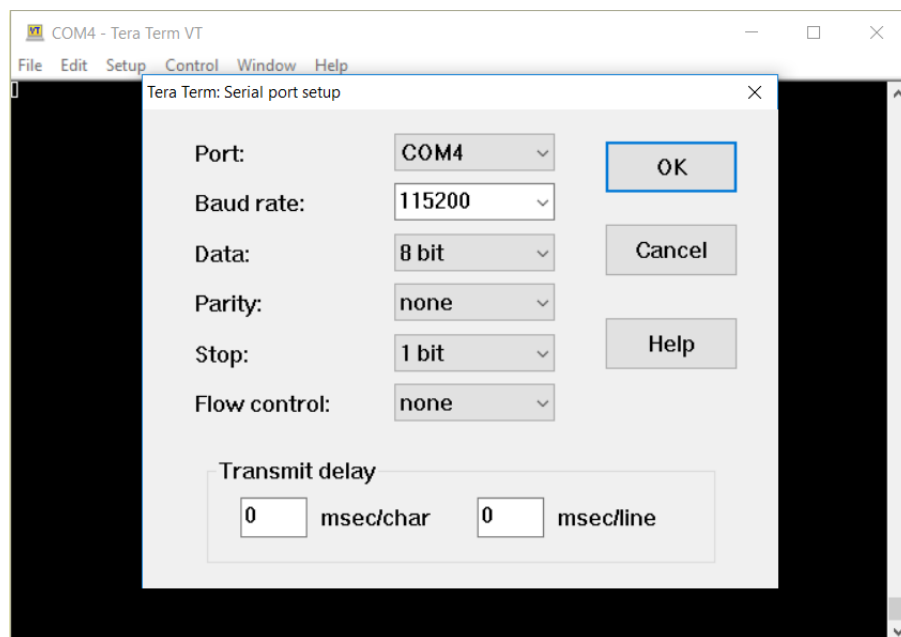


FIGURA 20: configuración de los parámetros de la conexión serie en *Tera Term*.

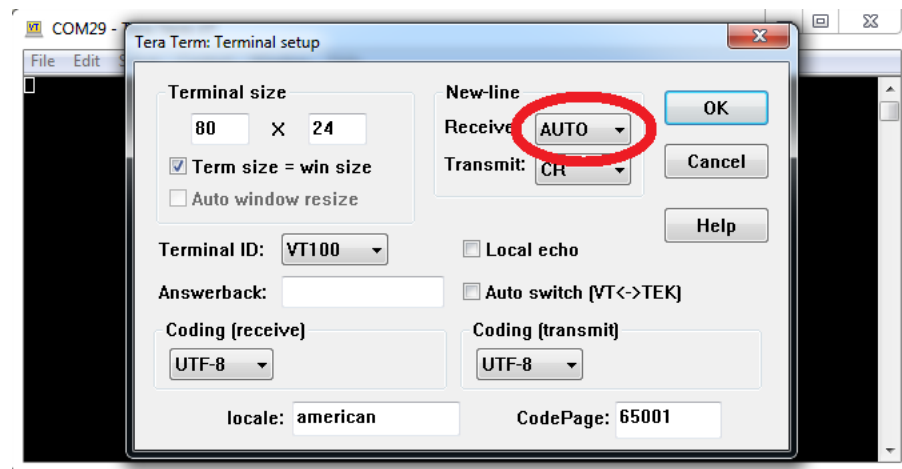


FIGURA 21: configuración para la conversión automática de "\n" a "\r\n" de los caracteres de fin de línea recibidos por *Tera Term*.

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

Copie el proyecto de *Keil μVision 5* del apartado 4.2.2 dentro de la carpeta MICR\P2\S3\10_printf 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 22.

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 3.2.3. En particular, cada `printf()` relacionado con la depuración se implementará como:

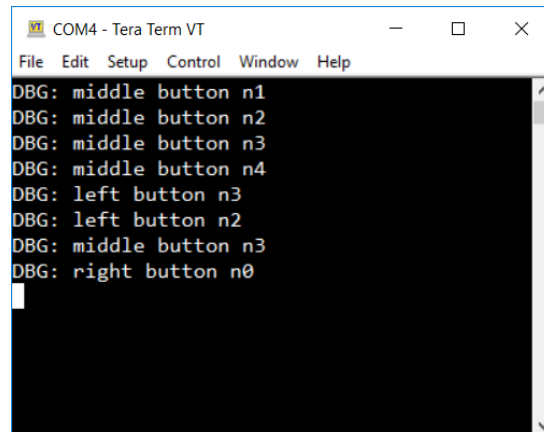


FIGURA 22: mensajes de depuración mostrados por *Tera Term*.

```
#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.

Deberá enseñar al docente, su programa generando los mensajes de depuración descritos anteriormente, cómo estos mensajes se muestran en la aplicación *Tera Term* y cómo su código permite eliminar todos estos mensajes de depuración empleando el mecanismo del `#define` recién descrito.

Tenga en cuenta que este mecanismo de depuración no sustituye, en absoluto, al más potente y flexible mecanismo de *breakpoints*, ejecución

paso a paso y uso de *watches*. Estos últimos mecanismos son claves, mientras que el uso de `printf()` para depurar es accesorio.

4.2.4. SUBIDA DE RESULTADOS A MOODLE

Elimine todas las carpetas `~build` y `~listings` de todos los proyectos de Keil μ Vision 5 de esta práctica. Borre también todos los ejecutables de ejemplo (`.bin`) que se han suministrado y, solo entonces, comprima las carpetas MICR\P2 y MICR\to_7seg en formato `.7z` (nivel de compresión Ultra). Suba este archivo comprimido a Moodle en el enlace correspondiente (una entrega por cada estudiante, ambos miembros de la pareja deberán subir el mismo fichero).

Terminan aquí las tareas a realizar para esta práctica.