

MICROPROCESADORES

PRÁCTICA 1

PRIMAVERA 2021 - 2022

MICROPROCESADORES

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

En este apartado se pretende emplear una aplicacion 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 extension es .uvprojx. Se abra 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, serí'a necesario realizar varias configuraciones del entorno de desarrollo, así como incluir la librería *mbed* en el proyecto.

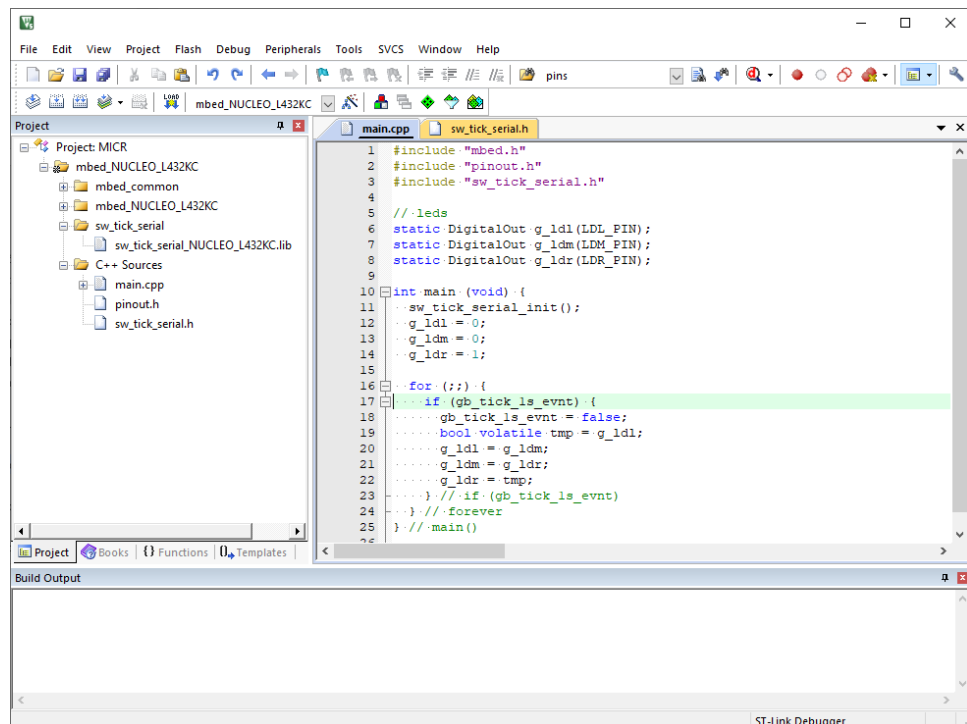


FIGURA 12: entorno de desarrollo *Keil μ Vision 5*.


Como recordara de lo dicho anteriormente y de las clases de teoría, en el caso de sistemas empotrados es muy habitual que el desarrollo de la aplicacion se haga en un ordenador (*host*) distinto al que va a, finalmente, ejecutar la aplicacion (*target*). En el caso que nos ocupa, el *host*

es el PC corriendo la aplicacion *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 ademas 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* comun a todas las tarjetas soportadas por esta librería.
- *mbed_NUCLEO_L432KC*. Parte de la librería *mbed* requerida unicamente por las placas *STM Nucleo-l432kc*.
- *sw_tick_serial*. Librería *sw_tick_serial*, de uso exclusivo durante esta practica.
- *C++ Sources*. Fuentes en C++ de la aplicacion.

La aplicacion final esta formada por los ficheros *main.cpp* y *pinout.h*. Una vez abierto el proyecto puede editarse el codigo que aparece en la parte derecha. Abra el fichero *main.cpp*, que es una sencilla aplicacion 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 como se relaciona este codigo con lo dicho en el apartado 5.4.2 en las paginas 14 y siguientes respecto a los pines de la placa *STM* a los que deben conectarse los LED (tendra que consultar tambien 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 extension *.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 ejecucion 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*.

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

Este proyecto esta 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 configuracion de las opciones de depuracion se realiza desde el menu Options for Target  dentro de la pestana Debug, como se ve en la figura 13. Observe en esta figura como la opcion 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.

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

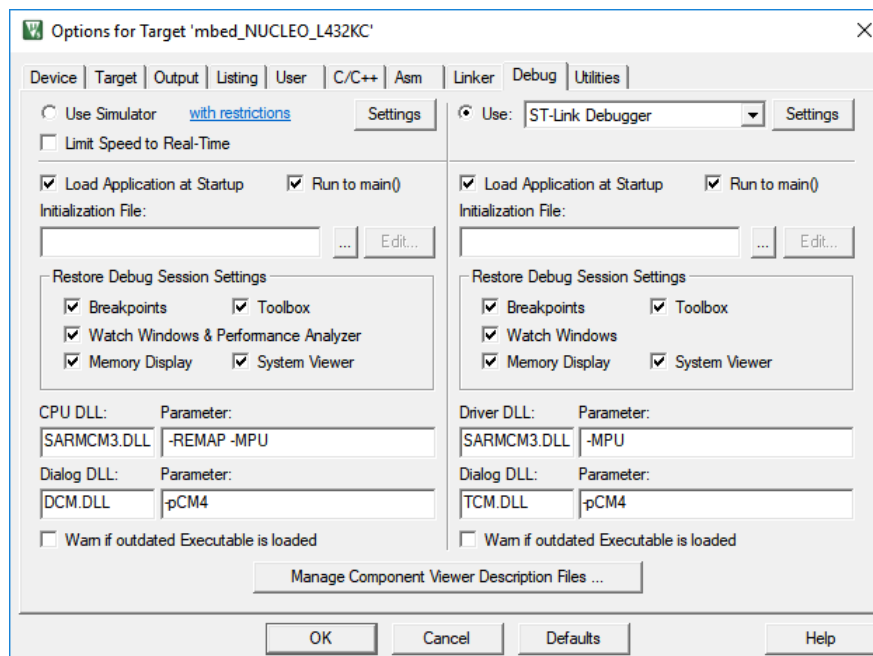


FIGURA 13: configuracion de depuracion para la placa mbed_NUCLEO_L432KC.

Tambien deben ajustarse algunos parametros del *debugger* accesibles desde Settings → Debug de la pestana Debug. Dichos parametros se encuentran descritos en la figura 14, destacados en rojo. Estos ajustes basicamente determinan que interfaz concreta de depuracion emplear (la interfaz SWD en vez de la JTAG), su frecuencia maxima de funcionamiento (en funcion del *hardware* disponible en la placa) y ciertas op-

ciones de conexion e inicializacion de la interfaz del *debugger*. Verifique que dichos ajustes son como los mostrados en la figura 14.

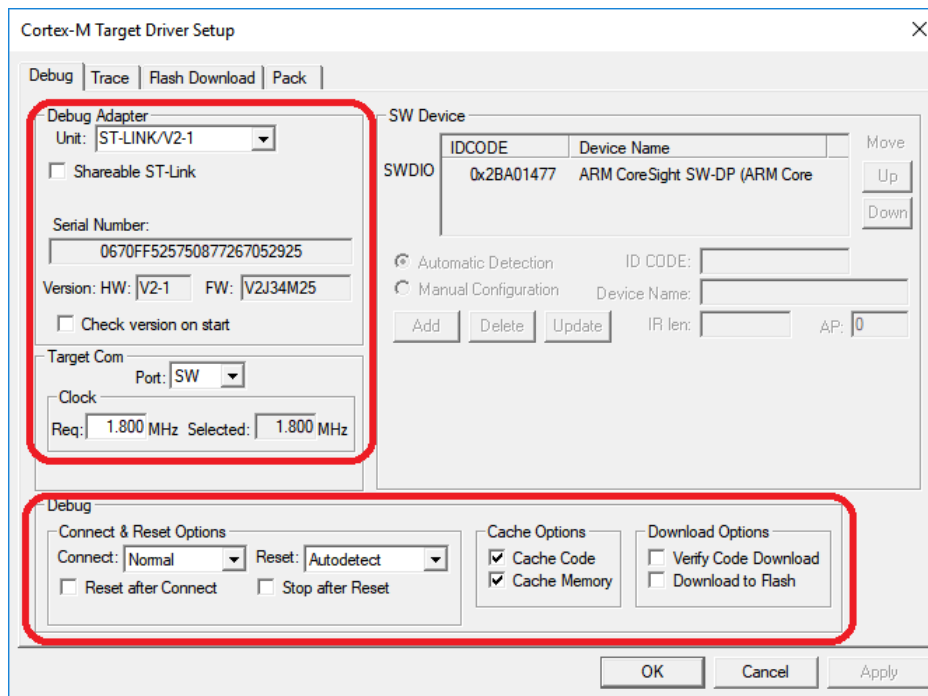



FIGURA 14: ajustes del *debugger* para la placa mbed_NUCLEO_L432KC.

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 permitira al depurador medir correctamente el tiempo de ejecucion de la aplicacion, tiempo que, durante la depuracion, se reporta en la línea de estado de la herramienta (la línea inferior de la ventana de *Keil μ Vision 5*, campo t1) y en la ventana de Registers, dentro del epígrafe Internal, campo Sec.

Por otro lado, tambien es necesario configurar el metodo empleado para volcar el ejecutable en la memoria *Flash* del microcontrolador. Para ello, en la pestaña Utilities de Options for Target , asegurese de que esta seleccionado Use Target Driver for Flash Programming y activas las opciones Use Debug Driver y Update Target before Debugging, como ve en la figura 16. Además, dentro del boton Settings de esa misma pestaña, y en la pestaña Flash Download debe seleccionarse el algoritmo concreto de programacion de la memoria *Flash* que emplea el procesador que se este usando, algoritmo que es STM32L4xx 256 KB *Flash*.

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

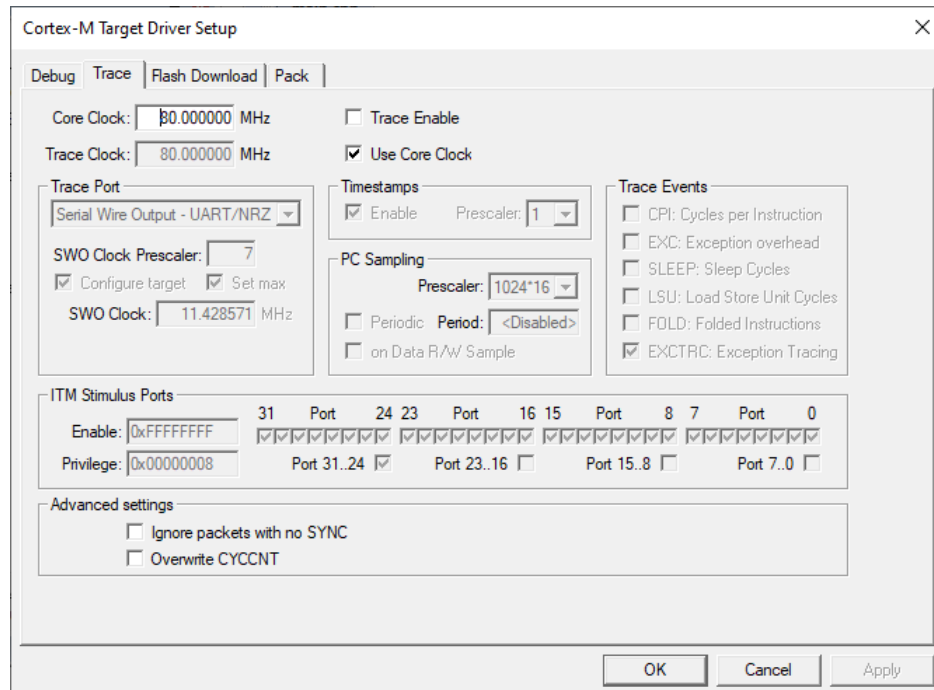


FIGURA 15: configuracion, para el depurador, de la frecuencia de reloj del procesador.

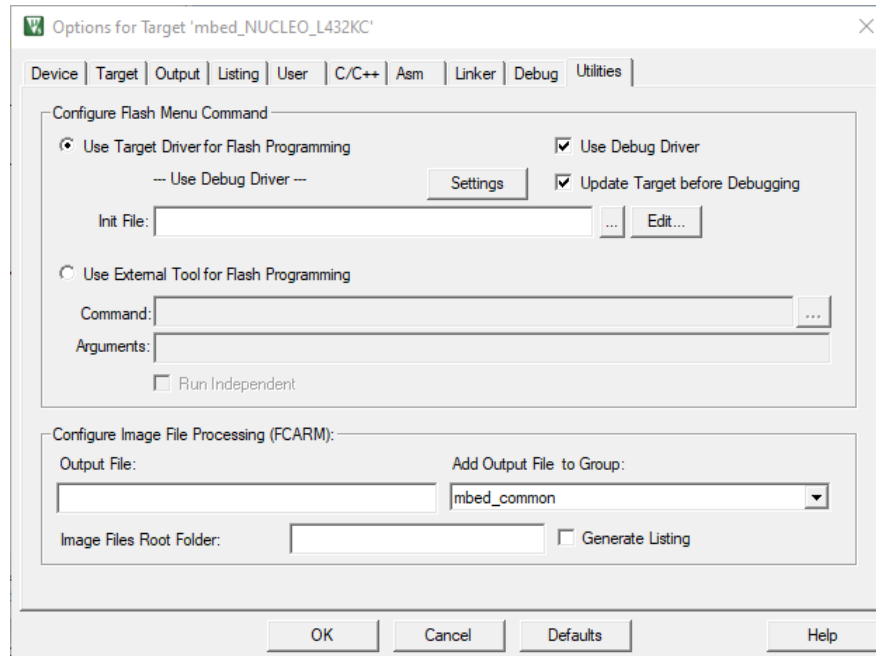


FIGURA 16: configuracion del mecanismo de volcado a la memoria *Flash* en Keil μ Vision 5.

Asegurese 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 µVision 5* le informara de errores al volcar el programa en la placa.

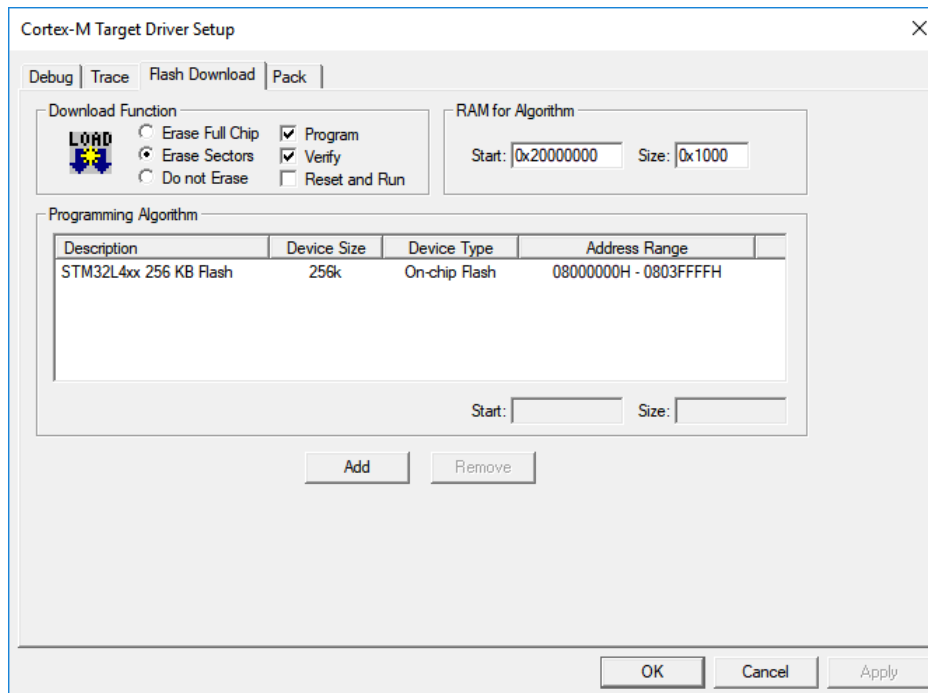













FIGURA 17: ajustes del algoritmo de programación de la *Flash* para la placa *mbed_NUCLEO_L432KC*.



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 .

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO 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 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 µVision 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 µVision 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 el 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*.

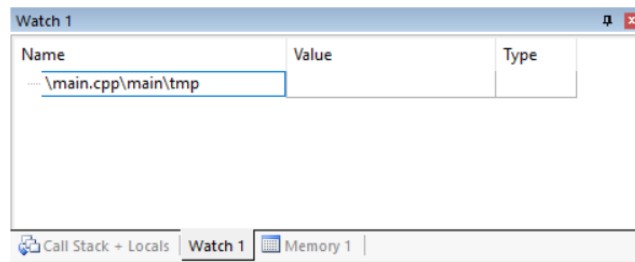


FIGURA 18: anadiendo una variable, mediante su nombre completo, a una ventana de Watch en Keil μ Vision 5.

Para este primer ejemplo:

1. compile el programa;
2. pase a modo de depuracion;
3. lance la ejecucion continua (F5), observe los LED;
4. ponga un punto de ruptura en la lí'nea 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 F7 —observando la evolucion de los LED y el valor de `tmp`— hasta la lí'nea 23 (`} // if (gb_tick_1s_evnt)`). En este momento lance la ejecucion continuada de nuevo (boton F5) hasta que vuelva a alcanzarse el *breakpoint*. Repita este proceso cuanto sea necesario hasta tener clara la evolucion 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 automaticamente a **true** una vez por segundo. Tampoco se preocupe, de momento, por la llamada a la funcion `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 ejecucion continua del programa (boton F5) 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 podr'ía ocurrir que el compilador determinase, buscando un ejecutable mas eficiente o rapido, 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 anadido el calificativo **volatile** a la definicion 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 almacenara siempre en memoria (y cada vez que se lea su valor, se leera de memoria, y cada vez que se modifique su valor, se escribira el nuevo valor en memoria) y podra ser accedida, durante la depuracion, 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 tambien 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 explorara mas adelante.

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

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

```

#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 seran de utilidad mas adelante.

5.2. Ejercicio 4 - Eventos

Para comprender completamente el funcionamiento del programa debera 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 interactuan con el exterior existe el concepto de *evento* (en ingles *event*). Un evento es cualquier suceso susceptible de generar una reaccion 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 reaccion 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 actualizacion de los LED y desactiva el *flag* a **false**, que volvera a activarse automaticamente al cabo de un segundo. El mecanismo por el cual este *flag* de evento se activa periodicamente sera el objeto de la practica 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 funcion `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 interes para esta sesion de la practica son los senalizados 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 periodicamente a **true** con el periodo indicado en su nombre.

Copie el proyecto completo de *Keil μ Vision 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.

5.3. 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 modulo 10. La cuenta debe actualizarse una vez por segundo. En la carpeta `MICR\P1\S1\E5` encontrara un fichero ejecutable `.bin` con la aplicacion ya compilada, de modo que pueda comprobar cual es el funcionamiento

esperado cargandola directamente en la placa, segun se describio en el apartado 5.4.3 en la pagina 17.

En la misma carpeta MICR\P1\S1\E5 encontrara tambien un proyecto de *Keil* con una aplicacion 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 µVision 5* preparado para trabajar sobre la placa STM.

5101 .ADICION DE GROUPS, FICHEROS Y LIBRERIAS A UN PROYECTO

Para realizar este apartado podra usar la librería *sw_tick_serial* que se ha descrito anteriormente y que encontrara en la carpeta MICR\sw_tick_serial. Dicha librería esta formada por dos ficheros: un .h (cabecera) y un .lib (objeto). Para incluir estos ficheros en el proyecto de *Keil µVision 5* se procedera de la siguiente forma:

1. Copie el fichero *sw_tick_serial.h* de su ubicacion original (MICR\sw_tick_serial) a la carpeta en la que esta el proyecto de *Keil µVision 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 abra un dialogo en el que se escogera la pestana 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 boton Add Files..., se abra el dialogo de inclusion 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 anadirse al proyecto tiene extension .h—. Seleccione el fichero *sw_tick_serial.h* y pulse Add y luego Close. Vera como el fichero *sw_tick_serial.h* queda anadido al *group* C++ Sources.
4. El fichero *sw_tick_serial.h* contiene la declaracion de los objetos que forman la librería *sw_tick_serial*, pero no contiene ninguna funcionalidad, esta se encuentra dentro del fichero .lib (objeto), que tambien deben ser incluidos en el proyecto. En este caso se va a crear un *group* específico para el que se llamara *sw_tick_serial*. En la misma pestana Project Items pulse sobre el boton de crear nuevo *group*  (en la parte central), de nombre *sw_tick_serial* al

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

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

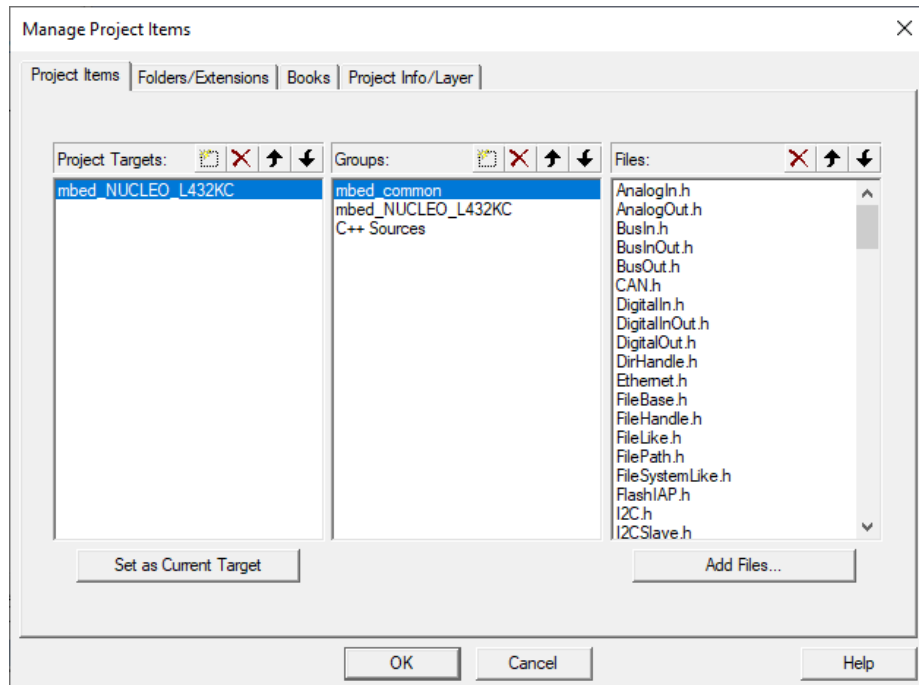


FIGURA 19: pestana Project Items.

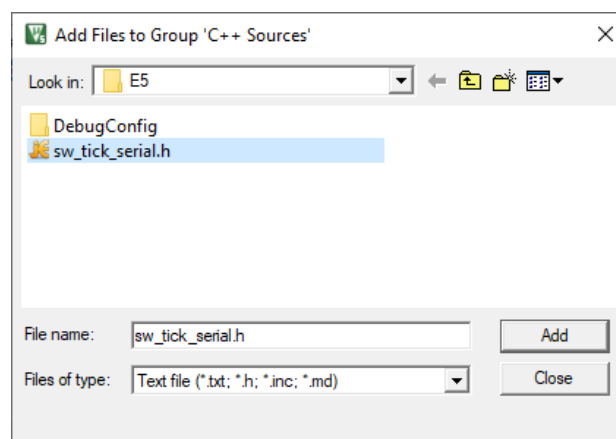


FIGURA 20: dialogo Add Files to Group.

5. Tal como hizo en el punto 3 anterior, anada 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 anade 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.

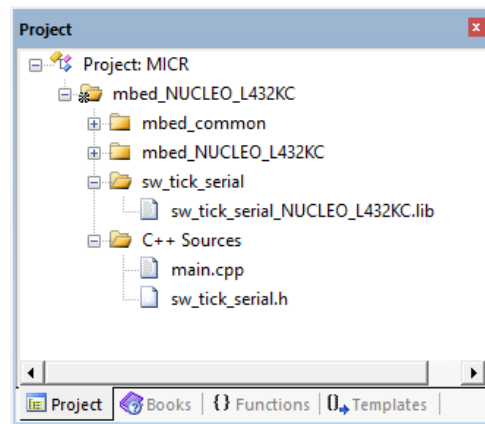


FIGURA 21: *group* y archivos de *sw_tick_serial* anadidos al proyecto.

Observe que, anadiendo la librería *sw_tick_serial* de esta forma, esta librería *no es copiada* desde su ubicacion original a la ubicacion 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 ubicacion relativa de la librería y del proyecto que la usa en el disco, de otra manera *Keil μ Vision 5* no podrí'a localizar la librería en el disco. Por eso en la seccion 4 de la pagina 7 se insistio 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 tambien en el proyecto (dentro del *group C++ Sources*) el fichero *pinout.h* que vio en apartados anteriores y que puede serle de utilidad.

5102 .REQUISITOS PARA ESTA APLICACION

Las 7 senales SGA a SGG que controlan los segmentos del *display* se agruparan en un *BusOut* (siendo SGA el LSB). El programa debera disponer de una variable que mantenga el estado de la cuenta (un numero del 0 al 9 que se incrementara una vez por segundo) y de una funcion capaz de traducir ese valor de cuenta en el codigo necesario a asignar al

BusOut para que en el *display* se muestre el símbolo asociado (ver figura 6 en la pagina 16). El prototipo de dicha funcion sería:

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

donde code es el valor de la cuenta y el valor devuelto por la funcion es el que hay que asignar al BusOut. Para implementar esta funcion se recomienda emplear una *tabla de búsqueda (lookup table)*. Se trata de un *array* de datos en los que cada posicion representa un codigo diferente, de modo que si se indexa el *array* con el valor 3, el codigo que se encuentra en esa posicion 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. Asegurese de que su funcion *to_7seg()* se comporta adecuadamente ante valores invalidos del parametro de entrada. En particular, si el parametro de entrada esta fuera del rango permitido (de 0 a 9), la funcion retornara el valor necesario para que se apaguen todos los segmentos del *display*. Tenga en cuenta que esta funcion 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.

5.4. 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

MICROPROCESADORES

el *display* de 7 segmentos la cuenta decimal ascendente de modulo 10 a una frecuencia de 1 Hz y que, simultaneamente, vaya «desplazando» a izquierdas un LED a una frecuencia de 10 Hz.

Se recomienda que copie proyecto del apartado anterior en MICR\P1\S1\E5 (cuidado, no incluya el .bin que se suministro) dentro de la carpeta MICR\P1\S1\E6 y que realice este apartado en esta carpeta. En la misma carpeta MICR\P1\S1\E6 encontrara 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 sesion presencial de esta practica.

6. TRABAJOS PREVIOS A LA SEGUNDA SESIÓN PRESENCIAL

Se describen a continuacion los trabajos a realizar antes de la segunda y ultima sesion presencial de la practica.

6.1. Ejercicio 7 - Montaje del circuito

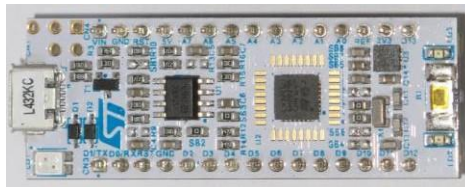
Se continua el montaje del circuito anadiendo ahora los recursos necesarios para la parte restante de esta practica.

6.1.1. MONTAJE DE PULSADORES Y SEGUNDO *DISPLAY*

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

Cada uno de los pulsadores gobernara una de las senales SWL, SWM y SWR (de *SWitch Left*, *SWitch Middle* y *SWitch Right*). Los pulsadores entregaran un nivel logico *bajo* al pulsarse. A su vez, las senales SWL, SWM y SWR se conectaran 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 Electronica I y Electronica II) debe incluirse una resistencia de *pull-up* para cada pulsador, sin embargo en este laboratorio se emplearan las resistencias internas de *pull-up* de los microcontroladores, por lo que no sera necesario montarlas.

Por lo que respecta al segundo de los *displays* de 7 segmentos, este se montara de forma que comparta las resistencias de limitacion con el *display* ya existente (en total habra 14 segmentos, pero solo 7 resistencias de limitacion de corriente) y tambien compartiran las senales SGA a

MICROPROCESADORES

SGG. Sin embargo, los catodos comunes de ambos *displays* se controlaran mediante transistores, de modo que, en cada instante del tiempo, solo uno de los catodos 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 numero pequeno de senales (tantas como segmentos tenga un *display* mas el numero de *displays*). Puede encontrar los detalles de esta conexion en el apartado 3.6.3 del libro de Toulson y Wilmshurst, en las clases de teorí'a o tambien en la *web*. Sin embargo debe tener en cuenta que en este laboratorio los transistores para la multiplexacion del *display* seran bipolares NPN (en el libro citado son MOSFET de acumulacion canal N) y, concretamente, seran de los tipos BC547 o 2N3904, a su eleccion. Cada transistor sera controlado por una de las senales DSL y DSR (de *DiSplay Left* y *DiSplay Right*), de modo que cuando una de estas senales se active a nivel *alto*, el *display* correspondiente se encienda, permaneciendo apagado en caso contrario. Las senales DSL y DSR se conectaran 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 Electronica I), sera necesario polarizar cada transistor adecuadamente para conseguir que se saturen o corten en funcion del nivel logico presente en la senal DSL/DSR que lo controle. Para ello debera calcular el valor de las resistencias de base que deben emplearse en dicho montaje. Los parametros necesarios para este calculo se encontraran en la *datasheet* de los microcontroladores (disponibles en *Moodle*) y del transistor concreto que haya empleado (cuya *datasheet* debe localizar usted).

En este apartado se cargara una aplicacion 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 multiplexacion) junto con el microcontrolador en las placas de prototipado, se volcara sobre la placa el ejecutable que se adjunta (carpeta MICR\P1\S2\E7). Este programa presenta un numero del 00 al 99 en el *display*. Si se pulsa el boton derecho, el numero se incrementa en una unidad; si se pulsa el boton izquierdo, se decrementa en una unidad; finalmente, si se pulsa el boton central, el numero mostrado (n) pasa a ser $99 - n$. A la vez, los LED mostraran, a una cadencia de un segundo, la cuenta binaria ascendente de 3 bits.

6.2. Ejercicio 8 - Gestión de los interruptores

Debera elaborar un programa que muestre, en el *display* de 7 segmentos de la derecha, la cuenta decimal ascendente de modulo 10, incrementandose la cuenta exclusivamente con cada pulsacion del pulsador izquierdo. A la vez, encendera uno de los tres LED que se encuentran en la placa e ira «desplazando» el LED encendido, hacia la izquierda, una posicion por cada pulsacion del pulsador derecho.

Segun lo dicho en la seccion 6.1, para conseguir encender el *display* derecho y apagar el izquierdo la senal DSR debera permanecer a nivel alto y la senal DSL a nivel bajo.

Para saber cuando se activan los pulsadores debe tener en cuenta que la librería *sw_tick_serial* que empleo en la anterior sesion 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 senales SWL, SWM y SMR, respectivamente. Es decir, estos *flags* se activan con cada pulsacion del pulsador correspondiente.

Se recomienda que para la realizacion de este apartado se parta del programa que se elaboro al final de la sesion anterior, modificandolo

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 sera el derecho, pero cada vez que se pulse el interruptor central, el *display* empleado conmutara *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 encontrara un ejecutable de esta aplicacion que puede emplear para contrastar su funcionamiento con el de su programa.

6.4. Ejercicio 10 - Trabajo con varios ficheros fuente

Una vez que este operativo el programa anterior, lo dividira en dos ficheros de codigo independientes (dos ficheros con extension .cpp) que ayuden a organizar el codigo 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 incluire exclusivamente la funcion empleada para la conversion a siete segmentos —`to_7seg()`— (fichero `to_7seg.cpp`), mientras que el otro contendra el codigo restante —incluyendo `main()`, fichero `main.cpp`—.

La funcion `to_7seg()` se modificara tambien para que, ademas de los dígitos del 0 al 9, muestre los 16 símbolos que se presentaron en la figura 6 de la pagina 16 (para valores del parametro de entrada desde 0 hasta 15). Como antes, para valores invalidos del parametro de entrada la funcion retornara el codigo necesario para apagar todos los segmentos.

Para realizar la separacion en varios ficheros fuente debe tener en cuenta lo explicado en la asignatura Programacion 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. Debera tambien recordar como anadir ficheros al proyecto de *Keil μ Vision 5* (ver la seccion 5.10.1), de modo que tanto `to_7seg.h` como `to_7seg.cpp` esten 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` debera incluirse tanto en `to_7seg.cpp` (obviamente) como en `to_7seg.h` —puesto que el prototipo de la funcion `to_7seg()` tambien 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 situacion 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`, `#ifndef`, `#ifdef` 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 codigo de un fichero fuente comprendido entre `#ifndef` NOMBRE_DE_MACRO y `#endif` no sera compilado si el macro (o símbolo) NOMBRE_DE_MACRO no ha sido definido anteriormente en ese fichero fuente;
- el codigo de un fichero fuente comprendido entre `#ifdef` NOMBRE_DE_MACRO y `#endif` no sera 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, ademas, 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`.

Debera 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 mas notables)*. Su programa debe evitar este tipo de comportamientos.

Para conseguir que en el *display* se muestren simultaneamente 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 pulsacion de un boton. Si ese cambio de *display*, en vez de hacerse al ritmo marcado por las pulsaciones, se hiciese periodicamente 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 numero) y cuando se enciende el izquierdo otro símbolo (la «n») el ojo percibiría la combinacion de las dos imagenes, 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 comoda es fijar un valor *mínimo* para esa frecuencia igual 50 Hz multiplicado por el numero de *displays* (o *campos*), en este caso, al haber dos *displays*, la frecuencia mínima de multiplexacion sería de 100 Hz. A frecuencias proximas 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 vision del observador).

La frecuencia de multiplexacion maxima viene fijada por la capacidad de los transistores de conmutar correctamente a esa frecuencia y porque el procesador sea suficientemente rapido para realizar la gestion del *display* a esa velocidad.

Se propone que se emplee para este ejercicio una frecuencia de multiplexacion de 1 kHz. Aunque puede parecer algo elevada, esta frecuencia es: sin duda superior a la frecuencia mínima; comoda 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 mas llamativas o no en funcion de los valores concretos que haya elegido para las resistencias de limitacion de corriente de los segmentos y de base de los transistores, así como del transistor concreto que se haya empleado. Tambien, a menor luz ambiente, mas aparentes son. Igualmente, desmontando el *display* de la derecha podra ver (con Sombras.bin) como se aprecia facilmente el numero que debiera presentarse en el *display* derecho, sobre la «n» del izquierdo.

realizar la gestion; pero suficientemente alta como para que puedan producirse sombras si existe una mala gestion del *display* por parte del programa*. En las siguientes practicas, en las que ya no se hara uso de la librería *sw_tick_serial*, podra emplear, si lo desea, frecuencias de multiplexacion mas bajas.

Debera emplear para este apartado su funcion `to_7seg()`, manteniendola 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 funcion equivalente a `to_7seg()`, pero escrita en lenguaje de ensamble, obviamente siguiendo el AAPCS.

Para anadir un nuevo fichero, dentro del *group* `to_7seg`, al proyecto de *Keil μ Vision 5*, pique con el boton derecho sobre el nombre del *group*, en la ventana izquierda Project y seleccione «Add New item to Group 'to_7seg'...», lo que abra el dialogo de adiccion 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 incorporara un nuevo archivo vacío `to_7seg.s` al proyecto en el *group* `to_7seg`, como se ve en la figura 23. Edite el fichero `to_7seg.s` anadiendo una funcion `to_7seg()`, pero ahora escrita en lenguaje de ensamble.

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

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

- hay instantes del tiempo en los que ambos *displays* estan encendidos simultaneamente;
- hay instantes del tiempo en los que el *display* que esta encendido esta mostrando informacion que no le corresponde a el.

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

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

debe compilarse y *linkarse* una de ellas, en este caso la de `to_7seg.s`. Para evitar la compilacion del fichero `to_7seg.cpp`, de modo que la funcion finalmente incorporada al ejecutable de la aplicacion sea la descrita en `to_7seg.s`, pique con el boton derecho sobre el fichero que desea eliminarse del proceso de compilacion `to_7seg.cpp` y seleccione la opcion «Options for File 'to_7seg.cpp'...», como se ve en la figura 24.

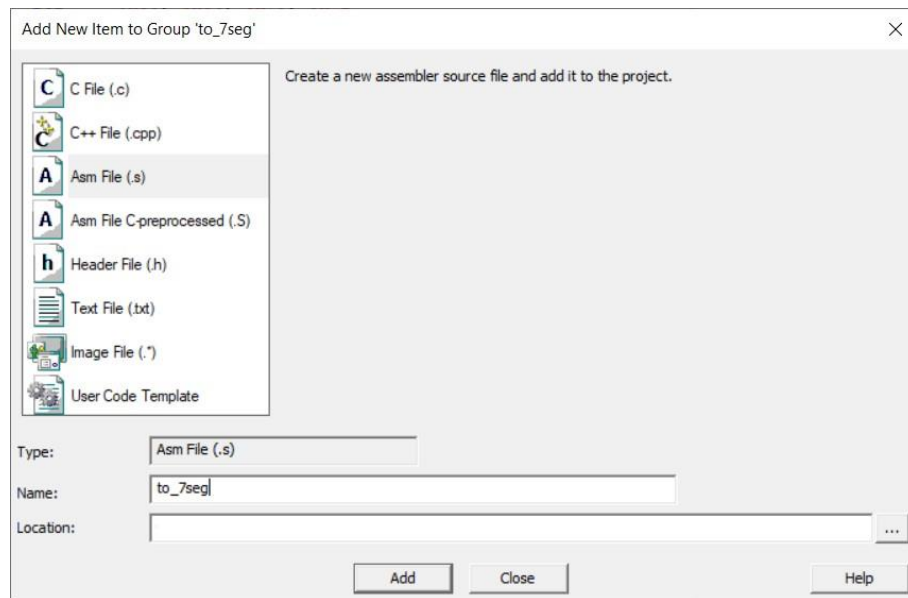


FIGURA 22: dialogo de adicon de nuevos ficheros a un *group* en Keil μ Vision 5.

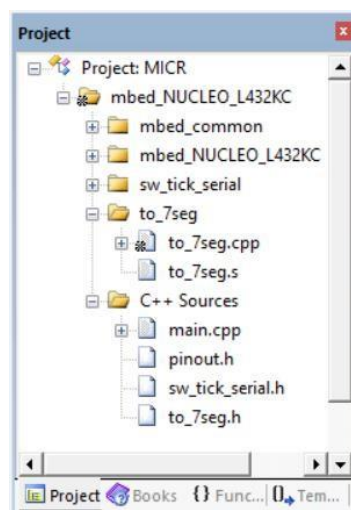


FIGURA 23: fichero `to_7seg.s` anadido al *group* `to_7seg` del proyecto.

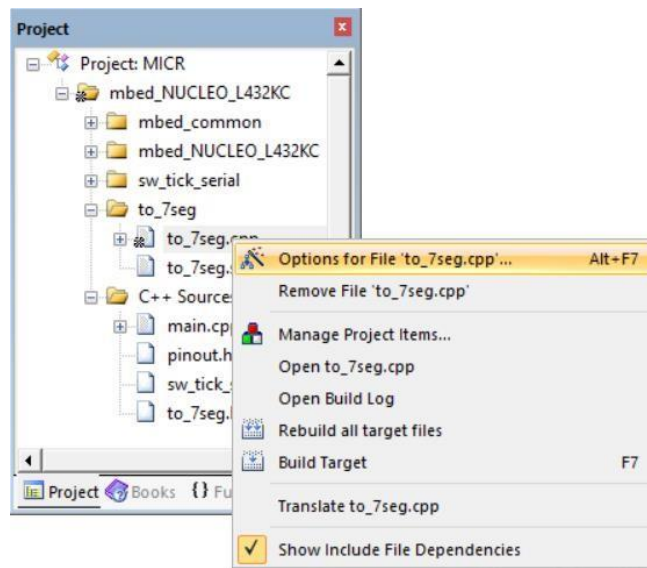


FIGURA 24: accediendo a las opciones del archivo to_7seg.cpp.

En el dialogo que aparece con las opciones del archivo to_7seg.cpp desmarque la opcion «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 interes si, por ejemplo, desea volver a la version en C++ de la funcion `to_7seg()` en vez de a la version en lenguaje de ensamble) y es accesible desde el IDE (*Integrated Development Environment*) de Keil μ Vision 5, pero no es compilado ni interfiere con la funcion del mismo nombre descrita en el archivo to_7seg.s. El hecho de que un fichero se encuentre excluido de la compilacion se muestra en la herramienta Keil μ Vision 5 mediante un pequeno s mbolo de prohibido junto al nombre del fichero, como se ve en la figura 26. En futuras actividades del laboratorio necesitara de nuevo excluir ficheros de la compilacion, de modo que tenga en mente los pasos que acaba de seguir.

Recuerde adem s que el fichero de cabeceras to_7seg.h tambien necesita ser modificado cuando la funcion cuyo prototipo describe esta escrita en lenguaje de ensamble pero va a ser llamada desde C++, como es este el caso.

Verifique el funcionamiento del programa corriendolo sobre la placa y compruebe, haciendo uso del depurador y de la ejecucion paso a paso, como la nueva funcion `to_7seg()` escrita en lenguaje de ensamble es ejecutada. Del mismo modo, cerciorese de que ha hecho un uso correcto

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

de las directivas **PROC** y **ENDP**, de modo que la nueva funcion **to_7seg()** aparezca en el *call stack* del depurador mientras esta siendo ejecutada.

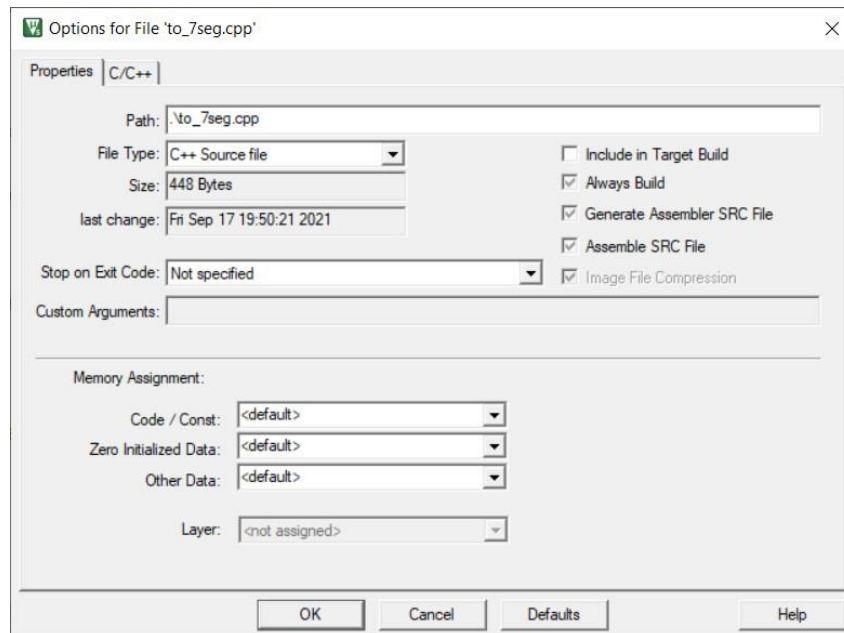


FIGURA 25: exclusion de un fichero de la compilacion en *Keil uVision 5*.

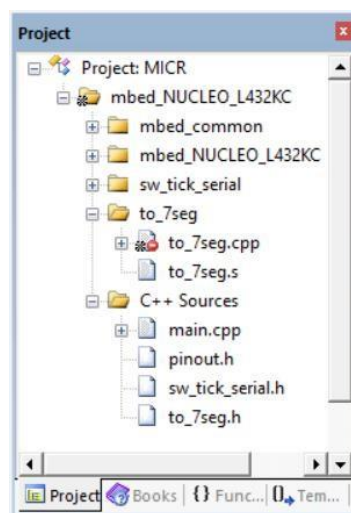


FIGURA 26: indicador de archivo excluido de la compilacion en *Keil uVision 5*.

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 este 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. Límitese ahora a configurar el programa terminal tal y como se describe. Más adelante entenderá el significado los parámetros que definen esta conexión.

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

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

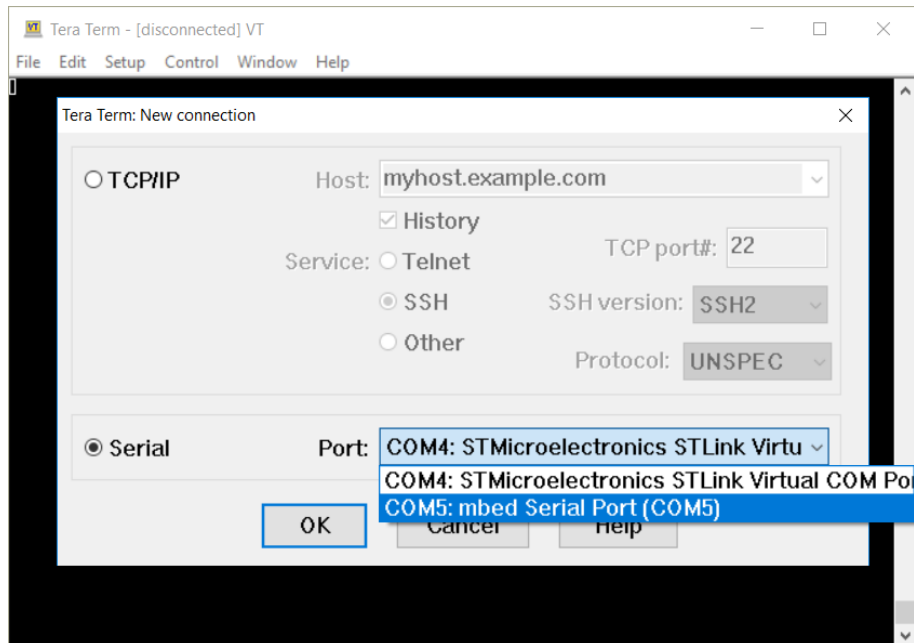


FIGURA 27: dialogo de nueva conexion de Tera Term.

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

Es necesario tambien configurar la forma en que el programa terminal procesara los indicadores de fin de línea enviados por la aplicacion. Mientras que usualmente se termina cada línea de un mensaje de `printf()` con un caracter de *cambio de línea* —line feed, "`\n`"—, la interfaz de Windows espera que cada línea se termine con una combinacion de dos caracteres de control: *retorno de carro* (*carriage return*) mas *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 (solo "`\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 automaticamente los "`\n`" recibidos en los "`\r\n`" esperados por Windows. Para ello, en el menu Setup → Terminal..., se establecera 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 validara con OK.

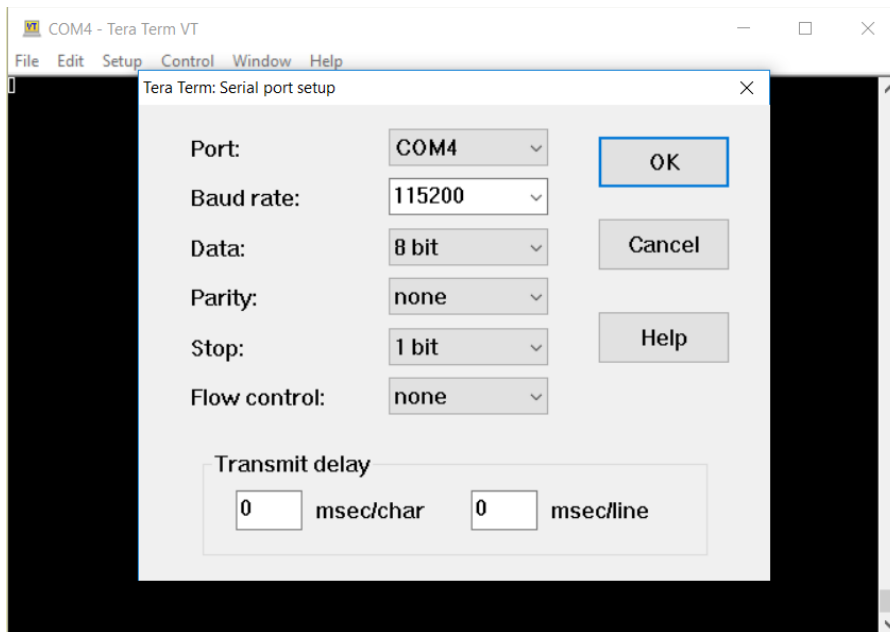


FIGURA 28: configuracion de los parametros de la conexion serie en *Tera Term*.

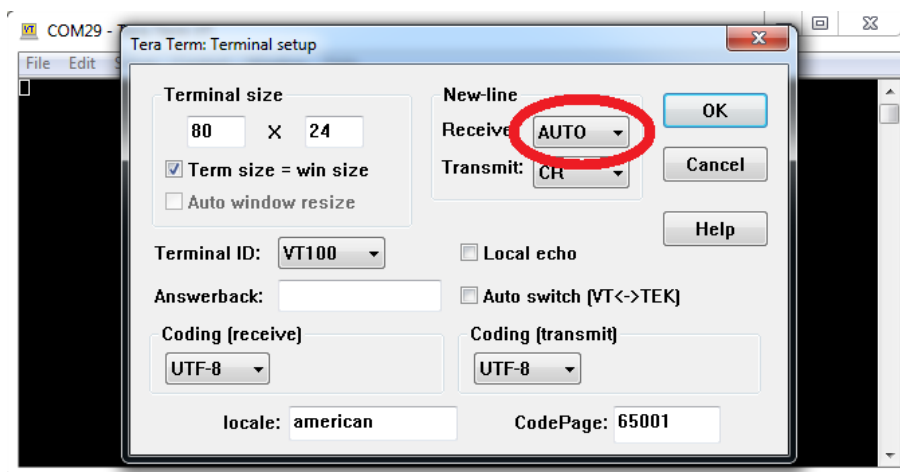


FIGURA 29: configuracion para la conversion automatica de "\n" a "\r\n" de los caracteres de fin de línea recibidos por *Tera Term*.

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

PRACTICA 1: ENTORNO, ENSAMBLE, I/O BASICO Y EVENTOS

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.

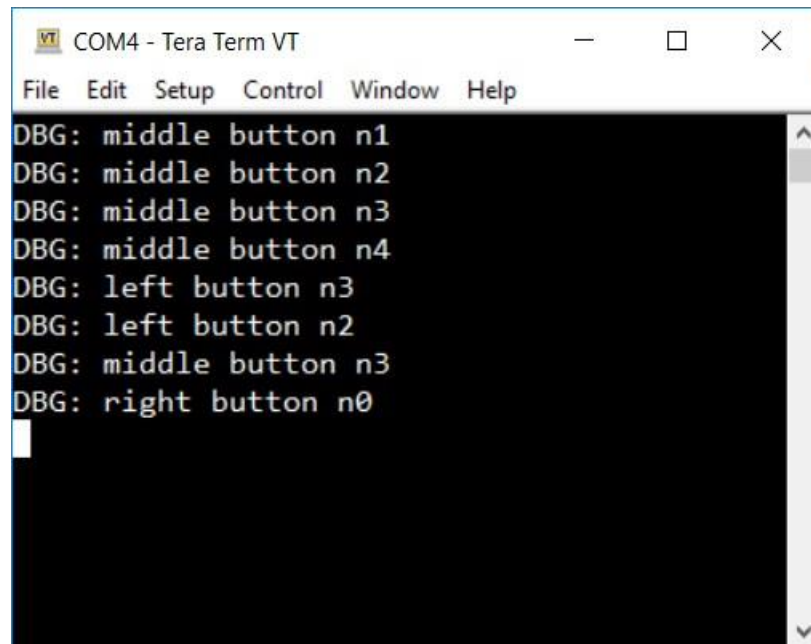


FIGURA 30: mensajes de depuración mostrados por *Tera Term*.

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, en absoluto, al más potente y flexible mecanismo de *breakpoints*, ejecución paso a paso y uso de *watches* y *call stack*. Estos últimos mecanismos son claves, mientras que el uso de `printf()` para depurar es accesorio. Aun 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. En la siguiente práctica no se hará uso de la librería `sw_tick_serial`, lo que da lugar a que el régimen binario del puerto serie sea el fijado por defecto por la librería `mbed`, solo 9600 bps, lo que incide aún más en la lentitud de `printf()`. En la última práctica ya dispondrá de los recursos necesarios para configurar según sus necesidades la velocidad binaria del puerto serie.

Terminan aquí las tareas a realizar para esta práctica.