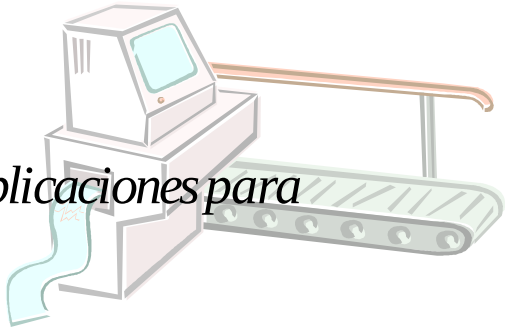
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Práctica 2

Desarrollo de código para microcontrolador

Laboratorio de Desarrollo de Aplicaciones para Sistemas Industriales



Desarrollo de código para microcontrolador

Introducción

En las dos últimas décadas los microcontroladores han cambiado radicalmente la forma en la que se analiza y controla el mundo que nos rodea. Es difícil imaginarnos nuestra vida sin teléfono, microondas, televisión, ascensor, lavadora, lavavajillas,… El núcleo del sistema electrónico de todos estos dispositivos es un microcontrolador. Para que estos elementos cumplan su función es necesario programarlos, indicar qué tareas tienen que llevar a cabo y como. C es un lenguaje que se puede usar para este propósito. Al ser un lenguaje de alto nivel, el trabajo del programador se simplifica. Una vez compilado y enlazado el programa fuente, se carga en la memoria del microcontrolador y ya se puede ejecutar.

En esta práctica el alumno va a trabajar con el entorno de desarrollo Keil µVision4. Este entorno se usara para desarrollar pequeños programas sobre microcontroladores de la familia ARM. El alumno también tendrá que usar el simulador incluido en el entorno, para comprobar el buen funcionamiento de los programas creados.

En los siguientes apartados se va a indicar los pasos que hay que seguir para crear un programa en este entorno, y simular su funcionamiento en el microcontrolador de la familia ARM.

Instalación del entorno

Desde el enlace https://www.keil.com/demo/eval/armv4.htm se permite la descarga de la versión de evaluación del entorno Keil µVision4. Esta versión permite crear programas de hasta 32 Kbytes, suficiente para la dimensión de las prácticas que se pretenden realizar.

¿Qué es µVision®4?

El entorno de desarrollo µVision© de Keil© es una herramienta IDE (Entorno de Desarrollo Integrado) profesional para el desarrollo de aplicaciones basadas en microcontrolador. Este entorno engloba una serie de componentes mediante los que se pueden llevar a cabo los procesos inherentes al desarrollo y depuración de aplicaciones. La figura siguiente muestra los módulos que integran este entorno de desarrollo.

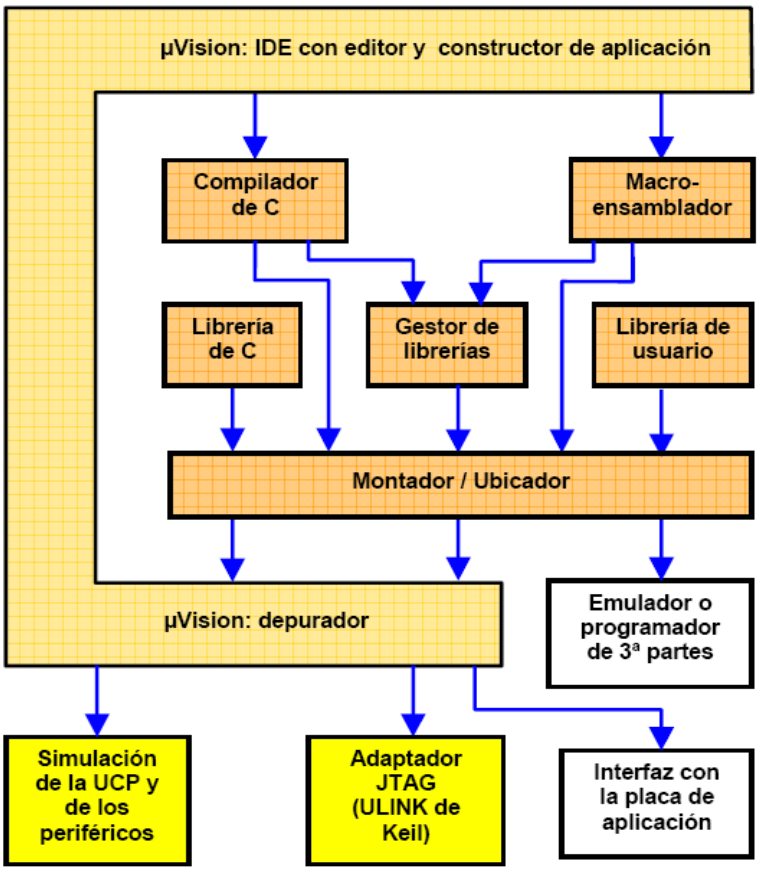


Figura 1. Componentes del entorno de desarrollo Keil µVision©.

La interfaz de este entorno nos permite definir las características de nuestro proyecto, como el tipo de procesador que utilizaremos, tipo de optimización a la hora de compilar, tipos de ficheros al ensamblar (o compilar) y montar, etc. µVision nos permite, asimismo, depurar el código desarrollado. Para ello, dispone del modo depuración, mediante el cual vuelca el código en la placa de desarrollo o placa de aplicación final, ejecutándose el código en esta y comunicándose con µVision. Para ello, el PC y la placa de desarrollo se conectan mediante un cable de conversión USB a JTAG. También es posible realizar la depuración sin un soporte físico o placa de desarrollo que ejecute realmente el código. Para ello, µVision integra un potente simulador con el que se puede simular el funcionamiento del procesador y todos los periféricos que éste integre.

Los ficheros fuente creados con el IDE de µVision se pasan al compilador o al macroensamblador para ser procesados y obtener los ficheros objeto reubicables, que luego pasan al módulo montador para obtener el fichero ejecutable. Los ficheros ejecutables (.HEX) se utilizan, por ejemplo, para programar la memoria Flash del microcontrolador.

El gestor de librerías permite la utilización de librerías de módulos objeto previamente generadas con el compilador o el macroensamblador. µVision también incluye una serie de librerías, por ejemplo, las que incluyen ciertas funciones matemáticas, etc.

Programación en C con KEIL µVision®4

A continuación se presenta una introducción al manejo del entorno de desarrollo µVision© de Keil©, una herramienta software para el desarrollo de proyectos en lenguaje C basados en

microcontroladores. Entre otras cosas, nos permite compilar, simular, depurar y cargar el código en nuestro microcontrolador (LPC1768, en el caso que nos ocupa).

Se hace una introducción a este entorno de desarrollo a través de un ejemplo sencillo que ilustra la creación de un proyecto, su simulación y depuración para visualizar variables, así como el empleo de los *breakpoints*.

Creación de un proyecto en C

La figura 1 esquematiza de una manera muy general los pasos que debemos seguir en el diseño de un programa para resolver un determinado problema con un microcontrolador.

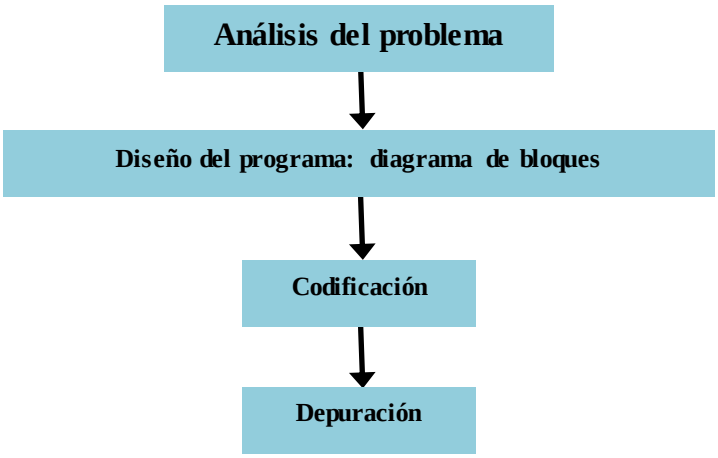
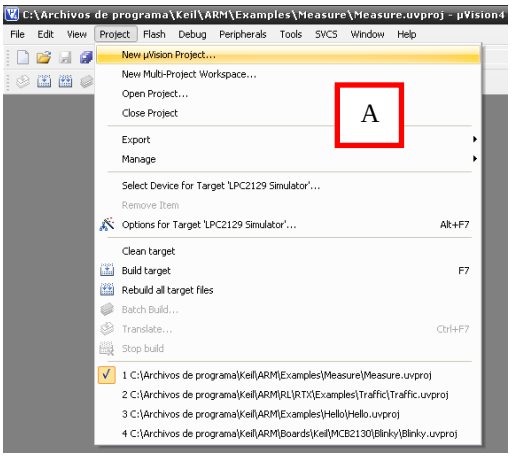
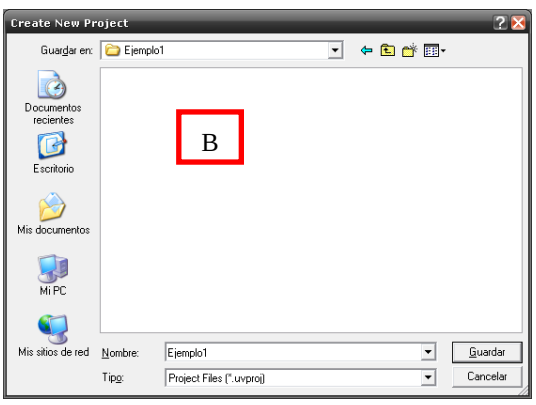


Figura 2. Pasos en el diseño de un programa para microcontrolador.

A continuación, se ilustrará cómo crear un proyecto en µVision con el programa fuente que previamente hemos escrito con un editor de texto o desde el propio editor de µVision. Luego se compilará este programa y se simulará para irlo depurando, si fuera necesario.

Comenzamos creando un nuevo proyecto, siguiendo los pasos indicados en la figura 3. Si ya se hubiese creado un proyecto anteriormente, entonces habría que abrirlo. A continuación se abre una ventana donde escribimos el nombre del proyecto (Ejemplo1) y lo guardamos (fig. 3B). Luego se abrirá otra ventana (fig. 3C) en la que hay que seleccionar el microcontrolador que se vaya a utilizar (LPC1768). Para ello se selecciona el fabricante y se hace clic en el símbolo '+' a su izquierda, con objeto de que aparezcan todos los modelos que corresponden a ese fabricante. En la siguiente ventana (fig. 3D) que se abre se pregunta si se desea copiar al proyecto un fichero con una plantilla para iniciar el código que se vaya a escribir (responder SI).





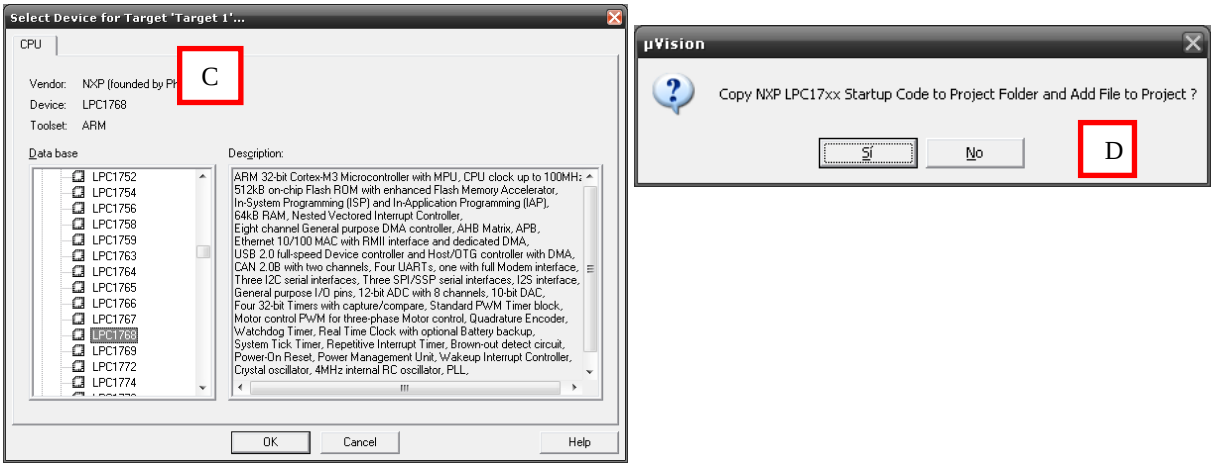


Figura 3. Ventanas para crear un nuevo proyecto, seleccionar el fabricante y el dispositivo.

El fichero *startup_LPC17xx.s* realiza una configuración inicial del microcontrolador, que permite al usuario trabajar con un nivel mayor de abstracción. Una vez realizada esta configuración, se llama a la función main(), que será la función principal de nuestro programa en C.

Configuración de opciones

Renombramos la entrada Target 1 como LPC1768 (haciendo click sobre ella, tras estar previamente seleccionada). Hacer click con el botón derecho del ratón sobre la entrada **LPC1768** para que aparezca su menú contextual y seleccionar **Options for Target 'LPC1768'…** (Figura 4).

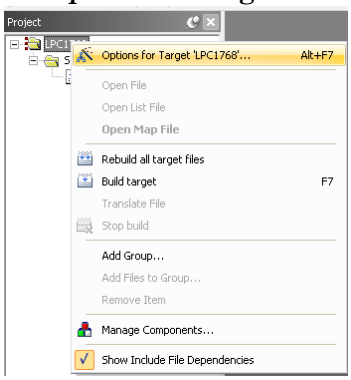


Figura 4. Menú contextual de LPC1768.

Una vez aparezca la ventana 'Options for Ejemplo1' configuramos las opciones que aparecen a continuación (figuras 5A, 5B y 5C). Seleccionamos la velocidad de cristal que viene por defecto (12 MHz), ya que será ese el valor del cristal de nuestra tarjeta en el resto de prácticas.

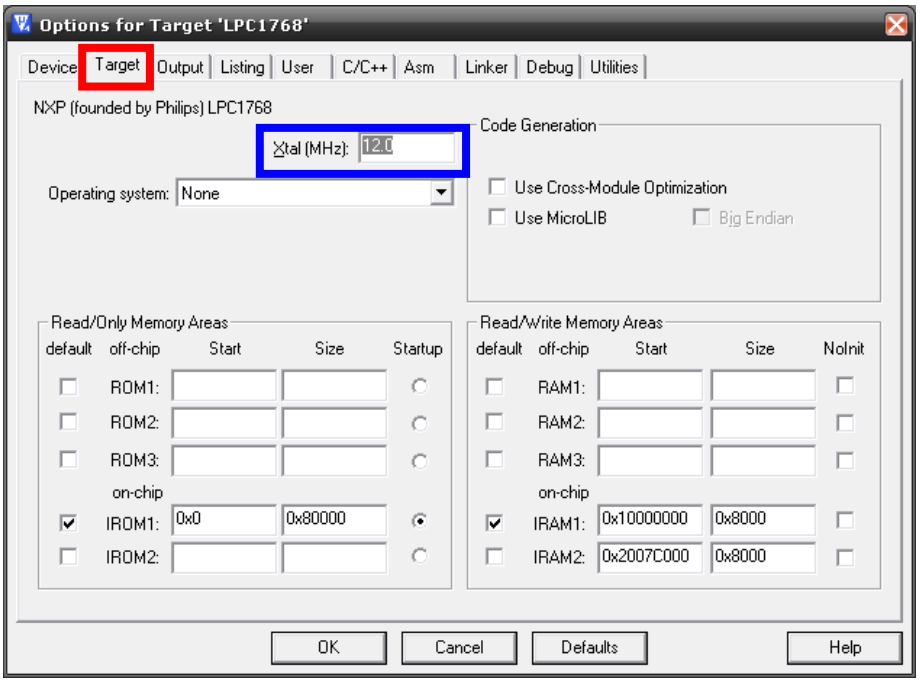


Figura 5A. Opciones para la pestaña Target.

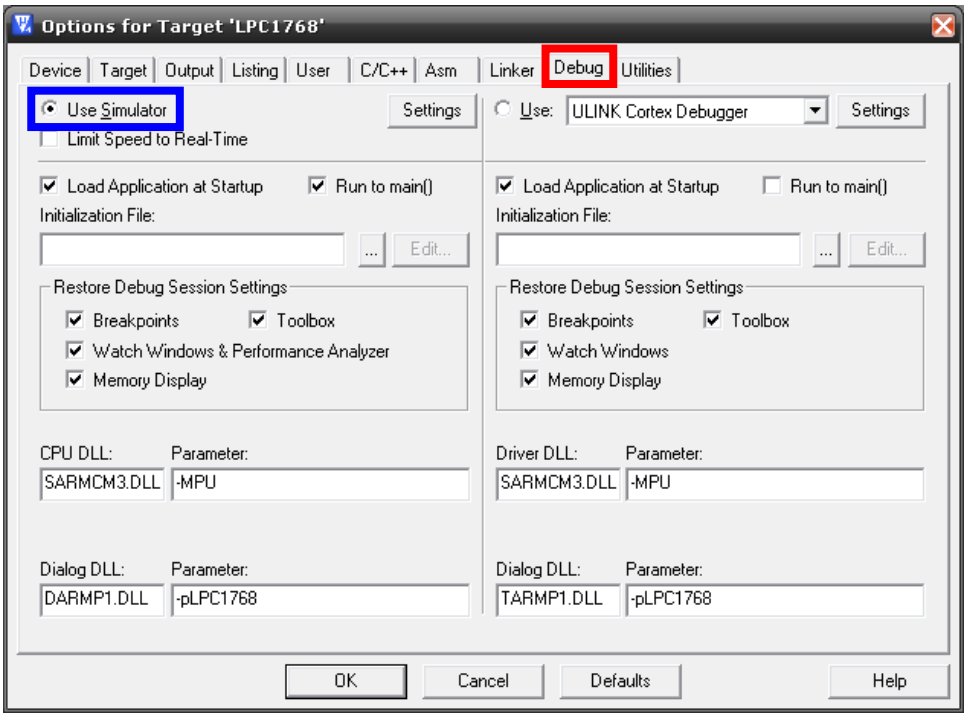


Figura 5B. Opciones para la pestaña Debug. Observar que esta práctica sólo será simulada, y no volcaremos el código sobre ninguna placa externa.

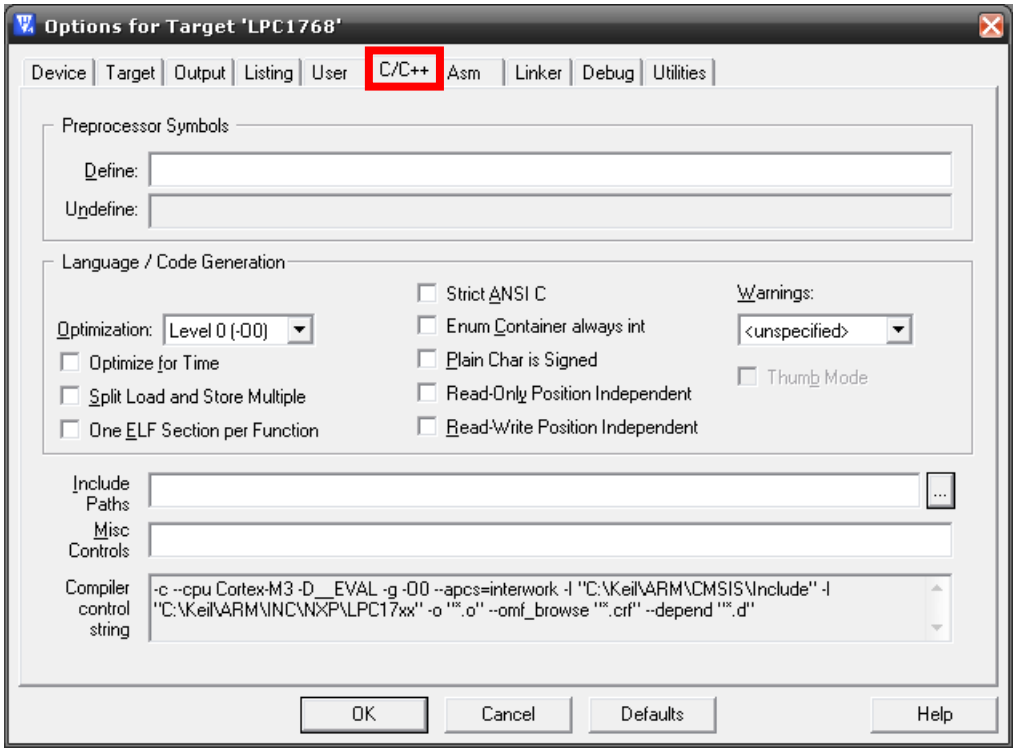


Figura 5C. Opciones para la pestaña C/C++.

Agregar ficheros al proyecto

Renombramos la entrada del árbol de proyecto *Source Group 1* como *StartUp*. Además de contener esta entrada de proyecto al fichero *startup_LPC17xx.s*, incluiremos otro fichero llamado *system_LPC17xx.c* necesario para la configuración inicial del microcontrolador. Para ello, proceder como se ilustra en las figuras 6 a 8 (*system_LPC17xx.c* se encuentra en la ruta C:\Keil\ARM\Startup\NXP\LPC17xx\, pero es muy importante copiar este fichero en el mismo directorio que nuestro proyecto e incluirlo desde ahí, para así no correr el riesgo de modificar el original). Ahora nos situamos sobre la entrada LPC1768 del árbol de proyecto, hacemos click con el botón derecho y seleccionamos *Add Group*…. Se creará una entrada llamada *New Group* que renombraremos como *SourceFiles*. Del mismo modo que hemos incluido el fichero *system_LPC17xx.c* a la entrada *StartUp*, incluiremos nuestro fichero fuente *main.c*, pero en la entrada *SourceFiles* que acabamos de crear (figura 9). Es necesario saber dónde hemos almacenado previamente los ficheros que deseemos agregar para así poder localizarlos.

NOTA: podemos editar previamente y guardar el fichero main.c con el listado de código que aparece en el siguiente listado.

```
void CalculaCuadrado(int *origen, int *destino, int n)
{
int i;
for(i=0;i<n;i++)
*(destino+i)=(*(origen+i))*(*(origen+i));
}

main()
{
static char var1;
int Tabla1[5]={2,3,9,12,15};
int TablaCuadrados[5];
static int i;

var1=0x12;
CalculaCuadrado(Tabla1, TablaCuadrados, 5);
while(1)
{
i=0;
i=1;
}
}
```

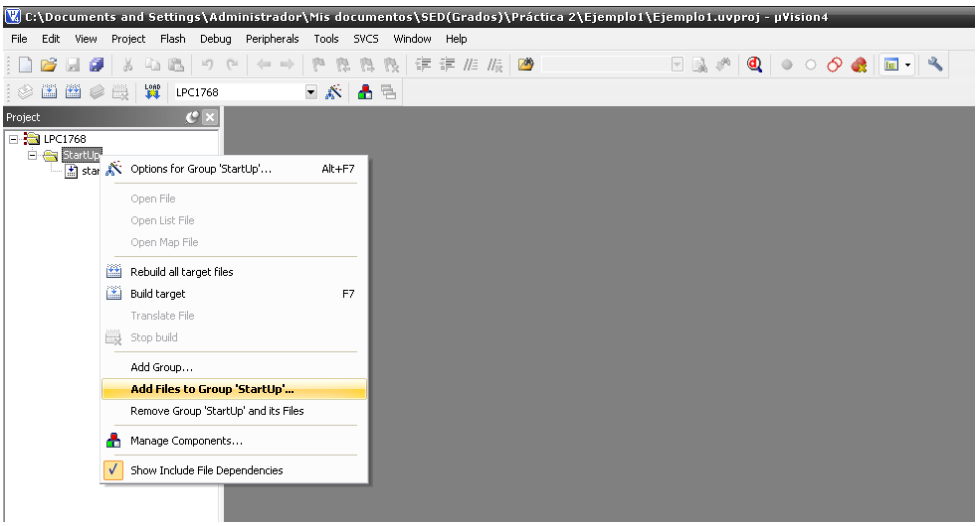


Figura 6. Menú contextual de la entrada 'StartUp'.

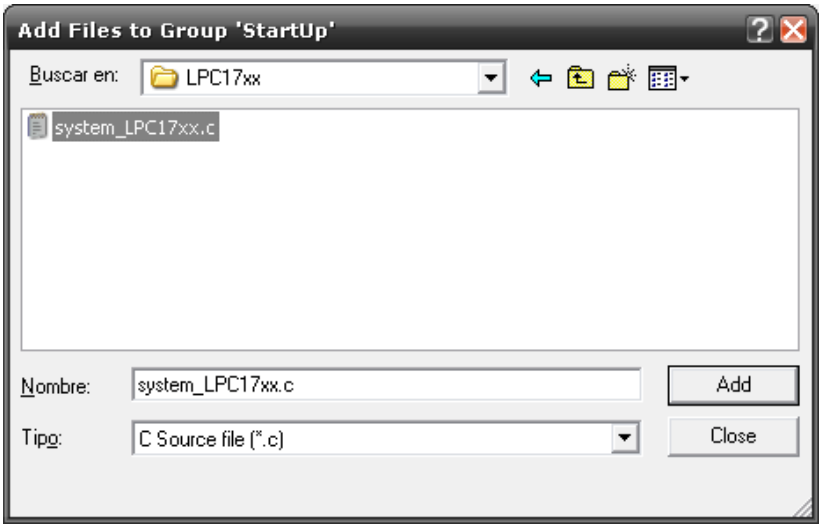


Figura 7. Fichero system_LPC17xx.c a ser agregado al proyecto.

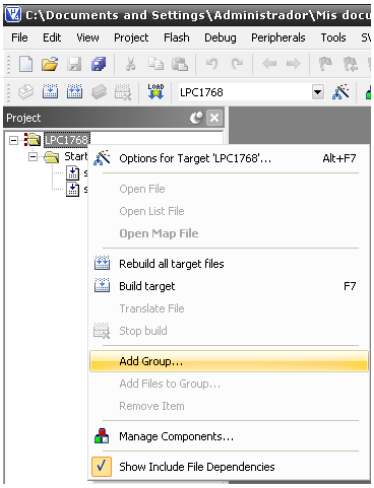


Figura 8. Agregamos grupo SourceFiles a la entrada LPC1768.

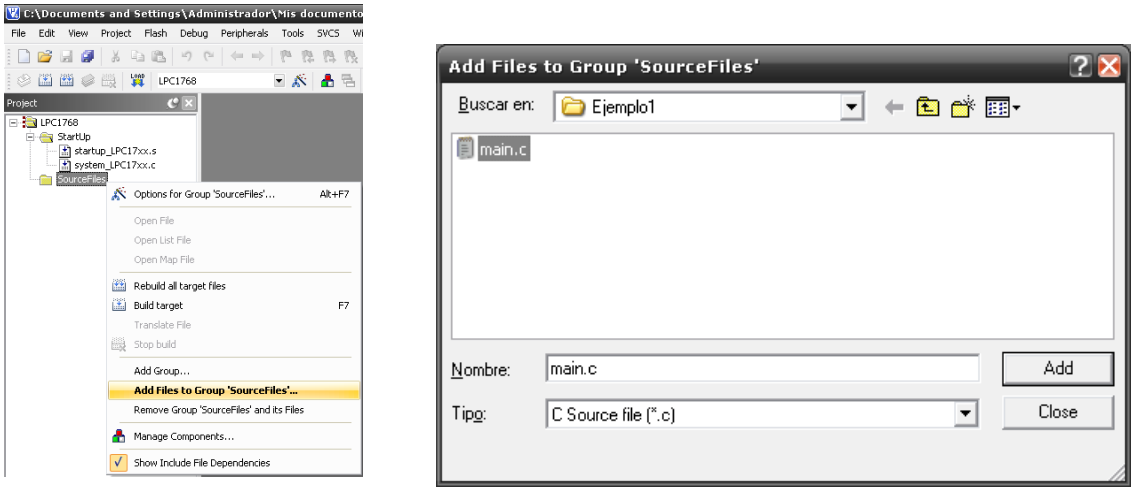


Figura 9. Agregamos main.c a la entrada SourceFiles.

En la figura 10 se muestra la estructura completa de nuestro proyecto ejemplo y el resultado de su compilación y enlazado mediante el comando Build (F7) o haciendo click sobre cualquiera de los iconos  Obsérvese la ventana inferior *Build Output* que nos indica si se ha producido algún error durante la compilación y también nos informa sobre el tamaño que ocupa nuestro programa.

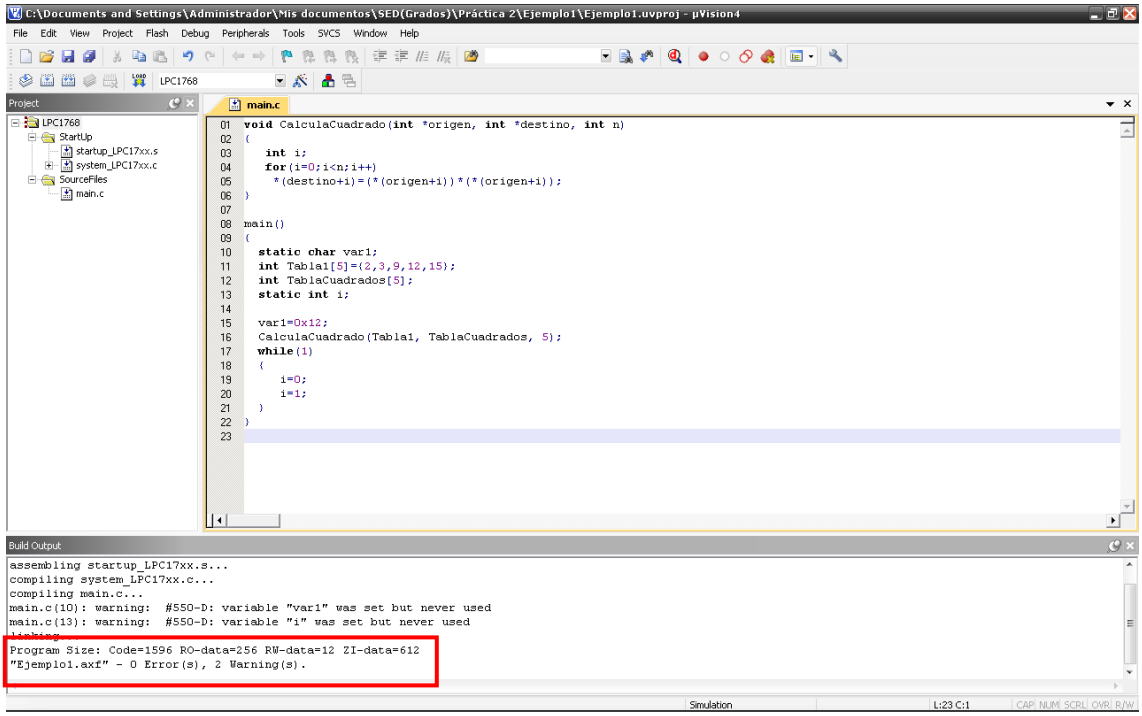


Figura 10. Compilado y "*linkado*" del proyecto (ZI-data: Zero Initialized Data, RO-data son constantes. Tamaño total de RAM = RW data + ZI data. Tamaño total de ROM = Code + RO data).

Simulación

Ahora tenemos la posibilidad de ejecutar el programa en el simulador que incluye µVision o de cargarlo a nuestra placa de desarrollo externa para que corra físicamente en el microcontrolador. De momento, haremos uso del simulador y veremos algunas de sus posibilidades interesantes. Para ello, arrancamos el simulador haciendo click sobre el botón  (F5) y aparecerá la ventana de la figura 10. Estos son algunos comandos que nos resultarán de utilidad:

- Ejecutaremos el programa paso a paso haciendo un click sobre  (F11) por cada instrucción.
- Si deseamos que se ejecute con un simple click toda una función, sin necesidad de ejecutar paso a paso cada una de las instrucciones que la componen, hacemos click sobre  (F10).
- Para ejecutar todo el código hasta la línea donde hemos colocado el cursor, click sobre el icono  (Ctrl.+F10).
- Para hacer reset y llevar la ejecución al comienzo del código, .
- Si lo que queremos es ejecutar todo el código de una vez,  (F5) y  para detenerlo.

A continuación se comentan algunas observaciones sobre las ventanas que aparecen en la figura 11:

- En la ventana *Disassembly* aparece el código traducido a ensamblador a partir del código fuente en C de las funciones *startup_LPC17xx.s* y *main.c*.
- En la zona de la izquierda aparece la ventana *Registers*, con el nombre y valor actual de todos los registros del microcontrolador. Se observa una marca grisácea sobre el/los registros que se ven modificados cada vez que ejecutamos un paso con Step(F11).
- Se puede apreciar una flecha o cursor de color amarillo que apunta siempre a la instrucción que se va a ejecutar (tanto en la ventana de ensamblador como en la de código c).

- También se muestra el tiempo transcurrido desde que se inició la ejecución del programa hasta el comienzo de la instrucción actual (parte inferior derecha).
- En la ventana *Locals* (parte inferior derecha) vemos el valor actual que van tomando las variables de nuestro programa.

En la ventana *Locals* también podemos comprobar que la variable *Tabla1* se ha almacenado a partir de la dirección de memoria 0x1000025C. Para mostrar el contenido de esta zona de memoria hacemos click sobre la pestaña *Memory 1* e introducimos en el campo *Address:* el valor 0x1000025C. Colocamos el cursor delante de la línea *CalculaCuadrado(Tabla1, TablaCuadrados, 5)* y ejecutamos hasta la posición del cursor (Ctrl+F10). En este momento ya está inicializada la variable *Tabla1* y en la figura 12 se puede ver el contenido de la zona de memoria donde se ha almacenado esta variable.

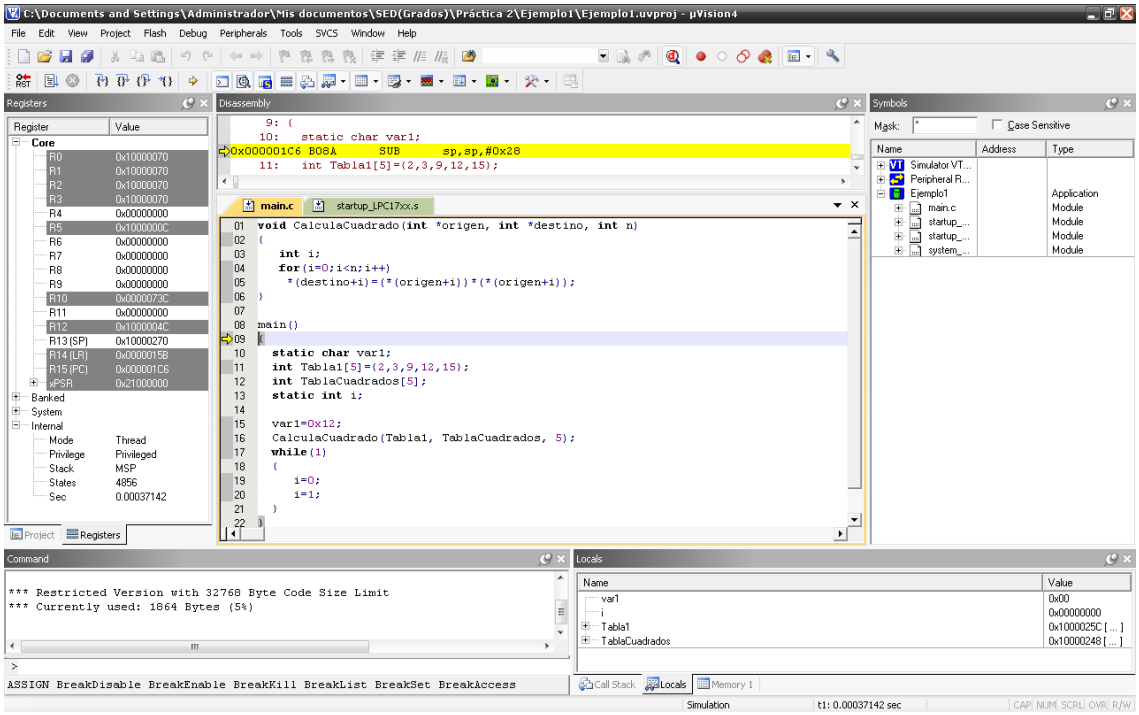


Figura 11. Ventana inicial del simulador.

Resulta interesante observar que cada valor de *Tabla1* ocupa 4 bytes porque se declaró como array de enteros. También se puede comprobar que el byte menos significativo se almacena en la parte más baja de memoria. Podemos modificar a mano el valor de una posición de memoria haciendo doble click sobre ella y editando su contenido.

Se puede visualizar el contenido de una variable determinada simplemente situando el cursor sobre ella en la ventana *Symbols* o en la ventana del código fuente (en este último caso, la variable debe pertenecer a la función que se esté ejecutando en ese momento).

Observe dónde se almacena el array *TablaCuadrados[5]* y compruebe cómo se actualiza mientras el programa se ejecuta paso a paso. Repita los mismos pasos con la variable *var1*.

IMPORTANTE: Para visualizar la variable *var1* es necesario hacer dos cosas: definirla como estática (*static char var1*) y en las opciones del compilador seleccionar *Optimization: Level 0*. Si no se selecciona este nivel de optimización, tampoco se ejecutará el bucle *while*, porque el compilador ve una cadena de operaciones sin mucho sentido: *i=0; i=1;* que las elimina del código compilado. Para visualizar la variable *var1* podemos hacer click con el botón derecho sobre ella y seleccionar *Add 'var1' to…* →*Watch1*, o bien escribiendo *&var1* en el campo *Address* de la ventana *Memory 1*.

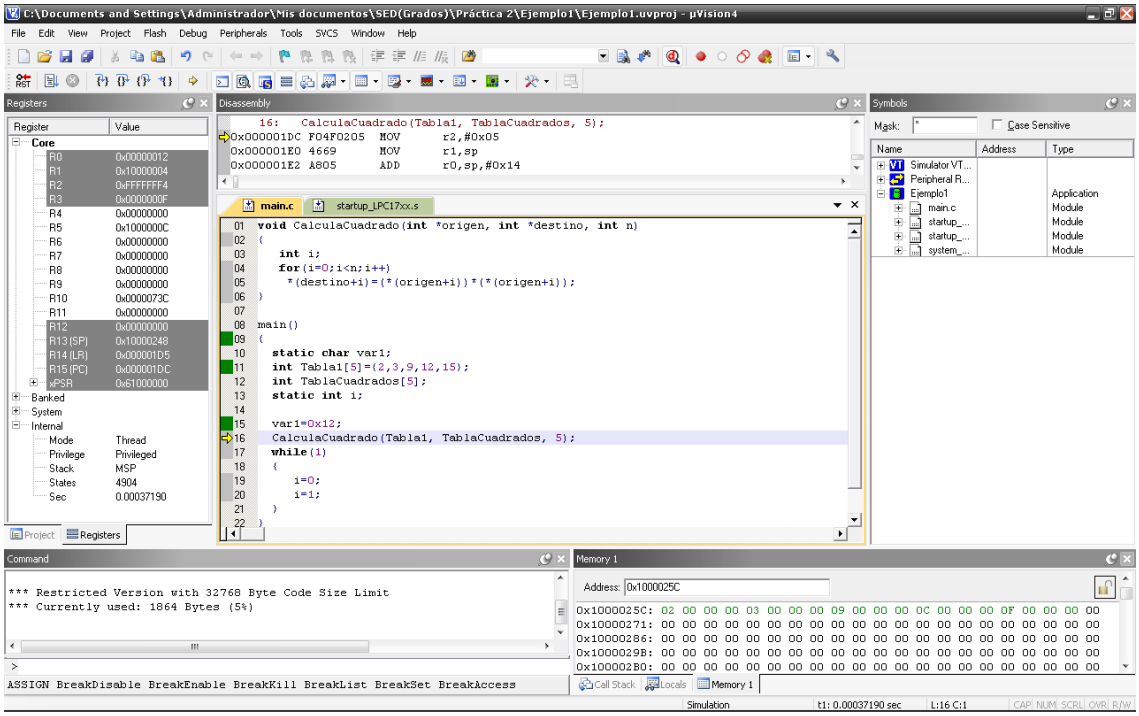


Figura 12. Contenido de la memoria donde se ha almacenado la variable Tabla1.

Utilización de Breakpoints en la simulación

Los *breakpoints* o puntos de ruptura son marcas que podemos colocar delante de las líneas de código para ayudarnos a depurarlo durante la simulación. Evitan, por ejemplo, tener que ejecutar paso a paso una porción de código que lleva mucho tiempo y sabemos a priori que funciona correctamente. De este modo, nos permiten ejecutar el programa y que éste quede detenido cuando se encuentra una línea que hemos marcado en el simulador con un *breakpoint*. A partir de ahí, podemos seguir ejecutando paso a paso para depurar cuidadosamente el código que sigue o seguir ejecutando el programa hasta que se encuentre el siguiente breakpoint, etc. Para insertar un *breakpoint*, sitúese en una línea de código y hacer click en **Insert/Remove Breakpoint (F9)**.

Practica Propuesta

Realícese un código que permita el cálculo en un microcontrolador LPC1768 de los números primos que se encuentran entre el 2 y el 65535. Los valores primos deben almacenarse en un array, que se debe declarar como variable global. Para eso, créese la función *uint8_t esprimo(uint16_t numero)* que retornará el valor 1 si el número que se le introduce como parámetro es primo o el valor 0 si no lo es.

```
#include "lpc17XX.h"
uint8_t esprimo(uint16_t numero){
…
…
return
}

main(){
…
esprimo(…);
…
}
```