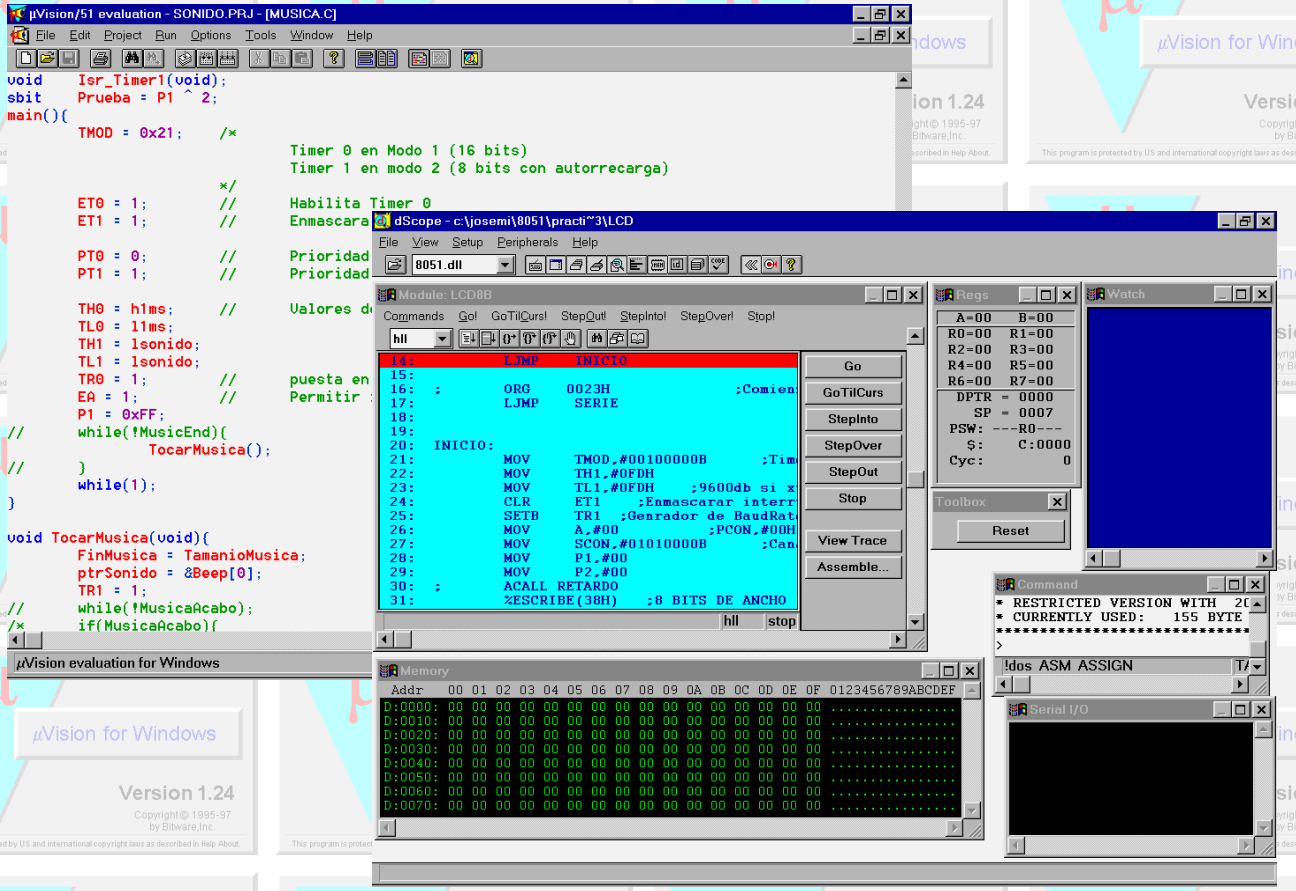
DEPARTAMENTO DE ELECTRONICA Y TELECOMUNICACIONES
ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA INDUSTRIAL E INGENIERIA TECNICA EN TOPOGRAFIA
UNIVERSIDAD DEL PAIS VASCO - EUSKAL HERRIKO UNIBERTSITATEA

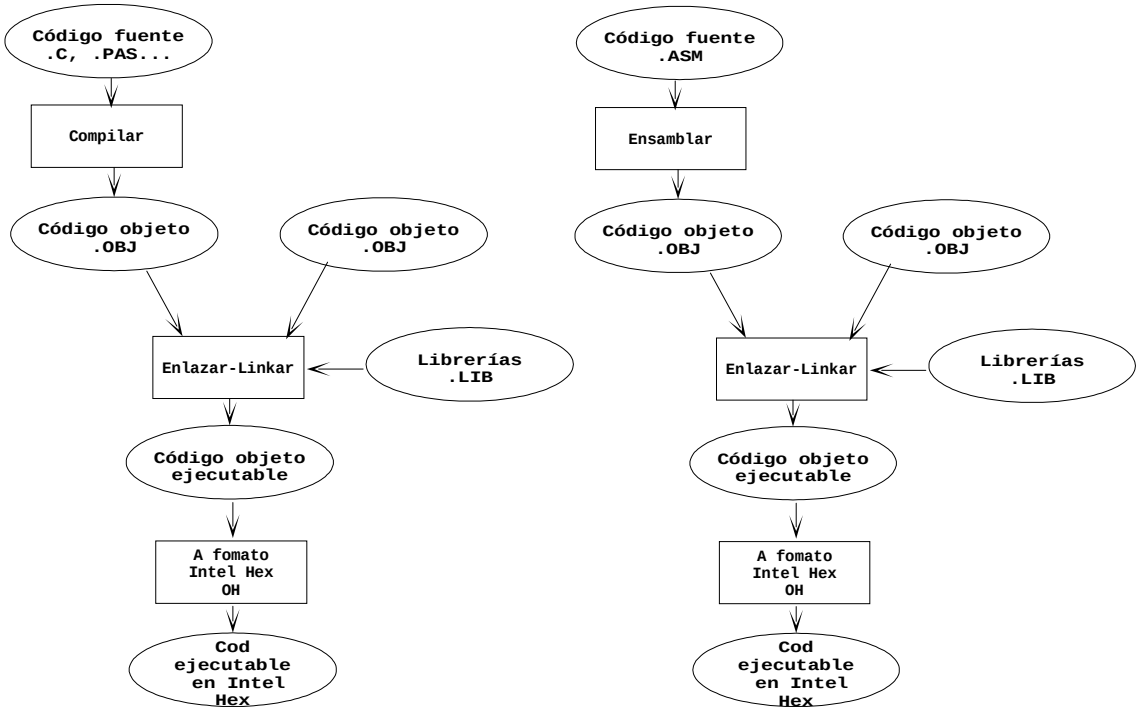
GUIA PARA EL USO DE µVision



J. M. GIL-GARCIA
MARZO 2000

Introducción

Cuando se programa un micro se parte de un fichero con un conjunto de instrucciones escritas bien en un lenguaje de alto nivel (C, Pascal,...) o bien con instrucciones propias del microprocesador con el que se está trabajando (lenguaje ensamblador). En ambos casos necesitamos pasar unas instrucciones escritas para ser entendidas por personas, a unos y ceros entendibles por la máquina. La forma de escribir los programas en ambos casos es mediante el uso de un editor de textos que sea capaz de salvar el programa en ASCII, sin introducir caracteres de control. Ejemplos válidos son el Notepad de Windows, el Edit de DOS o el PFE también para Windows. En todos los casos la forma de generar el fichero en formato Intel Hex, que es uno de los usados para grabar memorias, se puede ver en la siguiente figura.



Si trabajamos con lenguajes de alto nivel, el código fuente es compilado, mientras que si lo hacemos con los mnemotécnicos en lenguaje ensamblador, el código fuente es ensamblado. En ambos casos el resultado es un fichero en código máquina llamado código objeto. Para hacerlo ejecutable puede ser necesario enlazarlo (linkarlo) con otros módulos objeto y/o librerías que contienen funciones de uso habitual. El enlazador se encarga de recolocar todas las variables y rutinas en un fichero final de forma consistente. Llegado a este punto el fichero obtenido es ejecutable por el micro para el que ha sido compilado. Si este es un micro de la familia 8x86, su extensión será EXE (en el proceso de linkado se le añade una cabecera, además de unirse con los otros módulos fuente). Si es un microcontralodor, será un fichero binario que podrá ser ejecutado por él sin problemas, y que se deberá grabar en la memoria EPROM del sistema que diseñamos. A la hora de grabar la memoria, muchos programadores e incluso microcontroladores como la DALLAS 5000 entienden un formato que se denomina Intel Hex y cuya

estructura se explica en un apéndice de este manual. Para obtener ese formato se debe de ejecutar el programa OH sobre el fichero binario y se generará el archivo de extensión HEX con el mismo programa pero en ese formato.

En la siguiente imagen se puede ver la forma de usar los programas de ensamblado (ASM51), enlazado (RL51) y generación de HEX (OH) en la línea de comandos de DOS.

```
C:\8051\PracticaLCD>asm51 lcd8b.asm

DOS 7.10 (038-N) MCS-51 MACRO ASSEMBLER, V2.3
Copyright 1979, 1983, 1986 Intel Corporation

ASSEMBLY COMPLETE, NO ERRORS FOUND

C:\8051\PracticaLCD>rl51 lcd8b.obj


DOS 7.10 (046-N) MCS-51 RELOCATOR AND LINKER V3.2
Copyright 1982,1983,1986 Intel Corporation

C:\8051\PracticaLCD>oh lcd8b

DOS 7.10 (038-N)  OH  V1.1
Copyright 1986 Intel Corporation

C:\8051\PracticaLCD>
```

Tras el último paso en el directorio activo tendremos el archivo lcd8b.hex que será el que usemos para grabar la EPROM. Además del programa ejecutable, se generan o se pueden generar otros archivo como por ejemplo el .MAP que contiene una lista de las direcciones en que han quedado los diferentes módulos objeto tras ser linkados todos juntos; o el .LST que contiene un listado del programa fuente junto con la forma en que ha sido traducido a código máquina. Por si el proceso no fuera suficientemente complicado, cada uno de los programas admite una serie de switches o modificadores que cambian o añaden características al fichero generado final, en los que, de momento, no entramos.

El proceso de creación del archivo final implica el uso de, en este caso, cuatro programas: Editor, ASM51, RL51 y OH. Además si obtenemos errores durante el proceso de ensamblaje, estos salen en pantalla y sólo se reflejan en el archivo .LST, de tal forma que estamos obligados a tener varias ventanas abiertas para seguir los pasos de nuestros errores y poder subsanarlos rápidamente. Muchos fabricantes han optado por hacerle la vida más agradable (a un precio no necesariamente módico) al desarrollador diseñando entornos integrados donde, sin salir de la aplicación, se pueden realizar todas las operaciones hasta ahora descritas, con comodidad, haciendo transparente el proceso de creación del ejecutable final. Este es el caso de winIdea o µVision. Este último es la herramienta que se usará en clase para las prácticas, y el objeto del presente manual. Tanto la versión de evaluación de µVision como las herramientas mínimas para el desarrollo de programas con microcontroladores de la familia 51 se encuentran en la siguiente dirección del ftp de la escuela ftp://ftp.ehu.es/cidira/dptos/depjt/Programas/8051/compiladores/.

µVision

µVision es un entorno integrado de desarrollo para la familia del 8051 que no sólo nos da soporte para la edición y compilación de programas en C y en ensamblador sino que además dispone de un simulador de la arquitectura del 51 donde podemos depurar los programas.

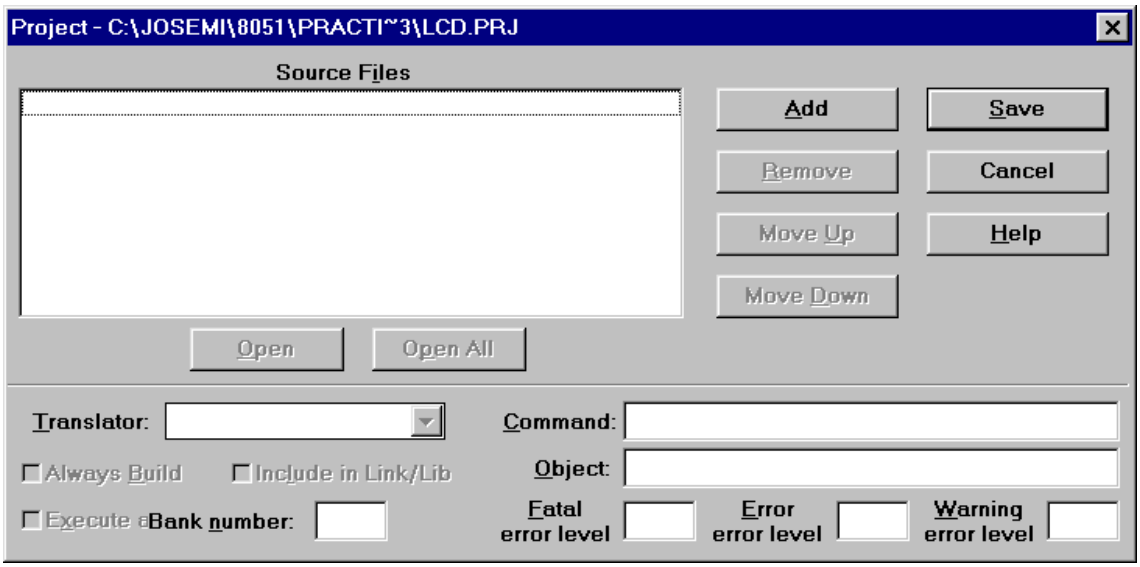
Instalar µVision

Si el programa se ha traído del FTP de la escuela se deberá de descomprimir en un directorio. Por defecto el programa viene en el directorio C51Eval, y ese es tan bueno como cualquier otro. El programa funciona en Win9x pero no aguanta nombres de archivos de más de 8 caracteres, ni que se copie en el escritorio, por lo que es mejor que cuelgue directamente del directorio raíz sin hacer experimentos con nombre largos o directorios raros. Si el programa se ha obtenido de la copia de CD del distribuidos, sólo hay que arrastrar el directorio C51Eval al C: y ya estará listo para trabajar.
Se recomienda crear un acceso directo al archivo Uvw51e.exe que se encuentra en C51Eval\bin y es el ejecutable de entrada al programa.

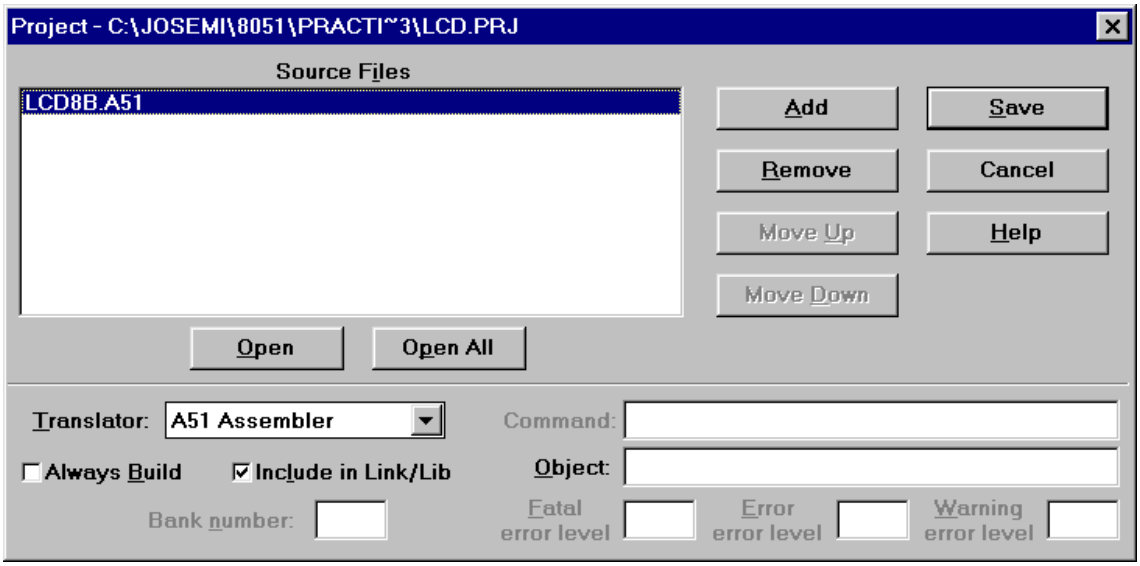
Crear un proyecto

µVision, como otros muchos entornos integrados, trabaja con proyectos. Un proyecto no es más que una relación de los ficheros fuente y librerías que formarán parte del ejecutable final, así como de las opciones de personalización que se elijan para ese proyecto. En otros entornos a esto mismo se le llama workspace o entorno de trabajo. Normalmente nuestro proyecto sólo va a constar de un solo archivo que será el que contenga el código. Cuando entramos por primera vez a µVision la ventana que se ve es parecida a la siguiente.

Lo primero que debemos hacer es crear un archivo con la extensión .A51 si es en ensamblador o .C si es en C y guardarlo con un nombre, por ejemplo lcd8b.a51. Para ello vamos a File->New y luego a File->Save As y le damos el nombre deseado. También se pueden usar los botones para estos menesteres. Una vez salvado debemos de crear el proyecto, que contendrá toda la información relativa al programa que se está haciendo. Para ello vamos a la opción Poject->New Project y tras dar el nombre del proyecto, por ejemplo LCD.prj, saldrá una ventana como la siguiente:

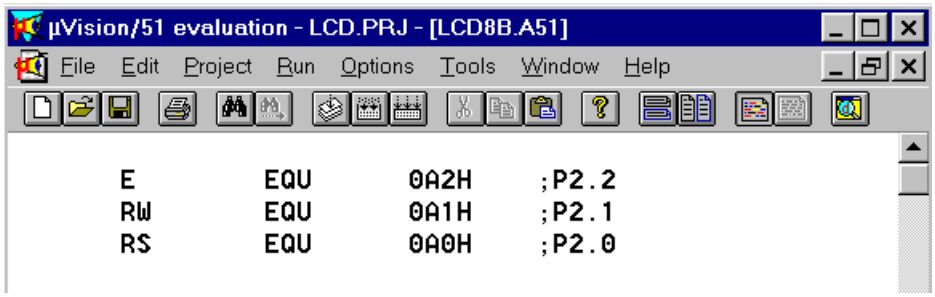


En ella debemos añadir (Add) el fichero fuente. Si trabajamos en ensamblador tendrá la extensión A51. Cuando hayamos terminado de incluir todos los ficheros fuente (C o A51) y librerías (.LIB) daremos a Close, y veremos la lista de archivos en Source Files.



Si damos al botón Open, se abrirá el programa seleccionado. Si no tenemos que incluir ningún fichero más daremos a Save y se cerrará la ventana.

Ahora en la barra del título se ve el proyecto con el que estamos trabajando así como el nombre del archivo que está siendo editado.



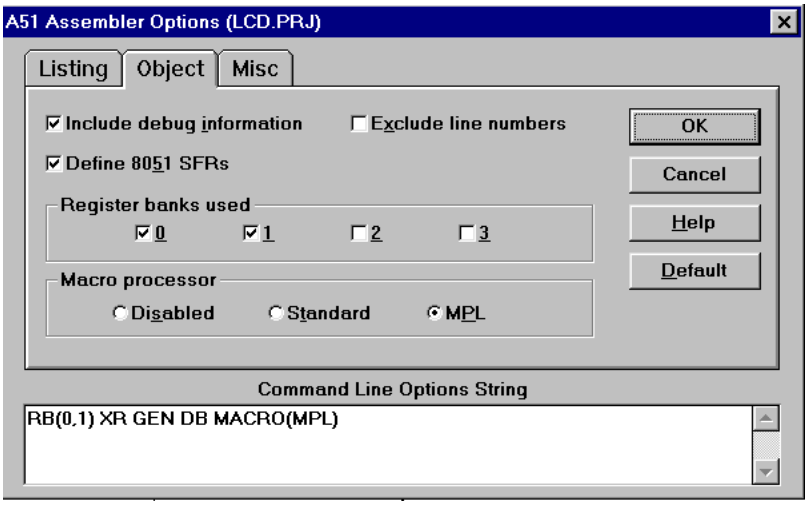
Si durante el proceso de creación del programa hay que añadir o quitar archivos al proyecto se deberá de ir a la opción Project->Edit Project. Si se quiere cambiar de proyecto habrá que cerrar el actual Project->Close Project y abrir el ya existente Project->Open Project. El programa se abrirá con el último proyecto con el que se ha trabajado por lo que siempre habrá que comprobar que el proyecto presente es el que se quiere que sea y no otro.

Fijar switches

Llegados a este punto y con todos los errores subsanados, ya podemos grabar el archivo final en la memoria o donde sea. Sin embargo, antes de lanzarnos a la tarea de ejecutar directamente el programa sobre el hardware real, es mejor simularlo antes 'en seco' para comprobar que funciona como nosotros esperamos que lo haga.

Para trabajar con comodidad en el simulador, habrá que hacer algunos cambios en la configuración del proyecto. Para ello abrimos la ventana Options->A51 Assambler y se seleccionan las siguientes opciones:

- en la pestaña de Listings, se seleccionan todas las opciones.
- En la ventana de Object se selecciona la casilla Include Debugger Information. De esta forma se podrá ver el propio código fuente en el depurador sin problemas. Si se usan Macros tal y como se explica en el manual de Intel se debe de tener seleccionada la opción MPL del apartado Macro Processor. Dependiendo del numero de bancos de registros que se usen se deberá de modificar la opción Register Banks Used

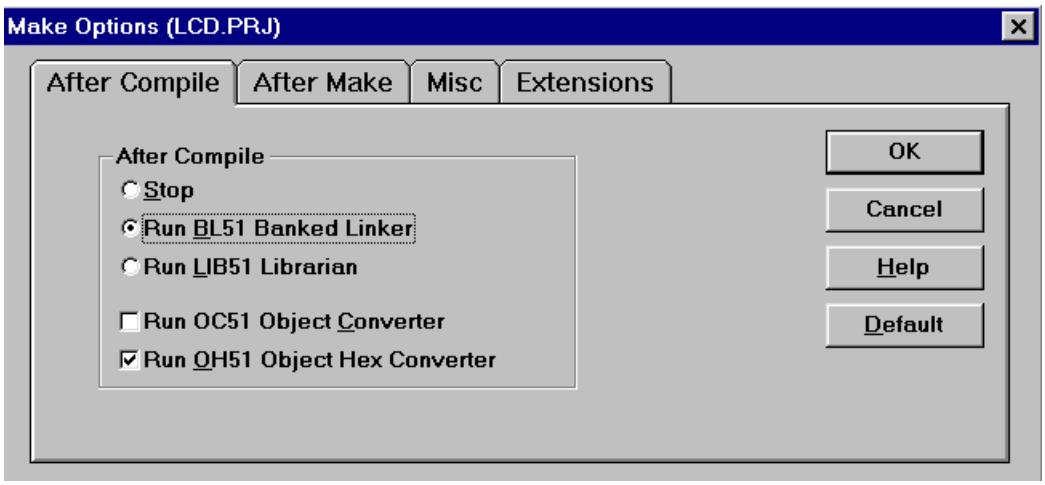


Si se trabaja en C se deberán de hacer algunos cambios en Options->C Compiler.

- En la pestaña de Listings se deberán de activar las cuatro primeras opciones.
- En la pestaña de Object se deberán de activar las 4 primeras opciones (aunque también la quinta si se quiere) pero sobre todo las dos correspondientes a la información del debugger. La pestaña Optimization puede ser crítica. Lo ideal es optimizar a nivel 6 favoreciendo bien tamaño, bien velocidad de ejecución, pero optimizar demasiado puede dar lugar a errores... Es mejor empezar sin optimizaciones (Level 0) y cuando hayamos acabado y sepamos que nuestro programa funciona, empezar a pasar niveles de optimización hasta llegar al 6 si no hay problemas (que no debería, pero en algunos casos los hay). La pestaña de modelo de memoria deberá de estar de acuerdo con lo que se haya decidido al escribir el programa (Variables internas o externas, etc.)

Tanto en la ventana de opciones para ensamblador como para C podemos ver en la opción Command Line Options String los modificadores que deberíamos usar para generar el programa objeto desde la línea del DOS o desde otro tipo de entorno que usara los compiladores del µVision, como PFE o win IDEA.

Por último podemos ver en Options->Make las diferentes opciones que tenemos a la hora de realizar el proceso. Aquí es donde determinamos que queremos hacer un fichero objeto y no una librería y que queremos el archivo final en formato Intel Hex.



Generar el archivo Hex

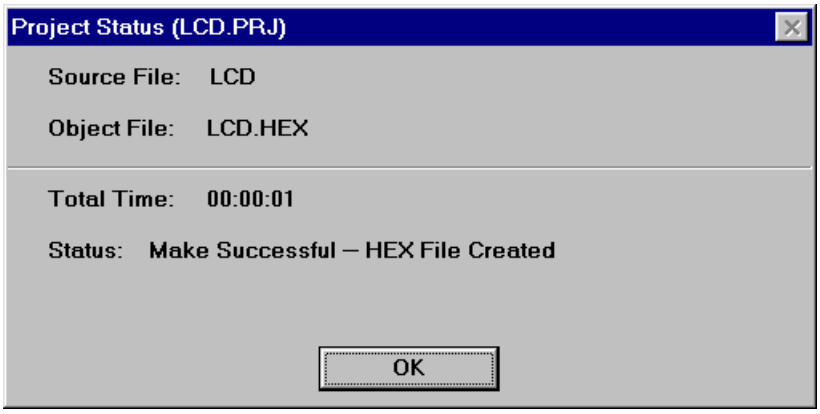
 Compile: Con esta opción de la barra de botones se compila-ensambla el fichero fuente para obtener el objeto. Pero nada más. No generaremos el fichero Hex únicamente con este botón.

 Update: Con esta opción se compilan-ensamblan aquellos ficheros que no han sido modificados desde la última vez en que se creó el programa final, y sólo aquellos. Después se linkan todos y se genera el fichero Intel Hex.

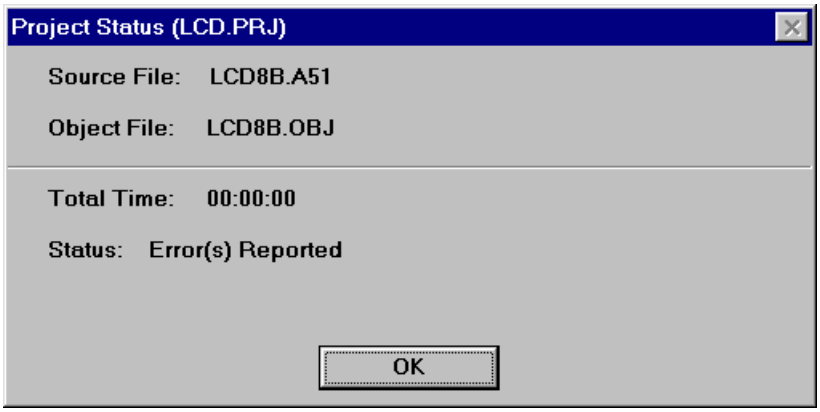
Build All: Con esta opción se compilan-ensamblan todos los archivos del proyecto, independientemente de si han sido modificados o no, y se enlazan para obtener el programa final con el formato Intel Hex.

Las dos últimas opciones son similares. Para proyectos grandes o máquinas lentas, escogeremos la segunda opción, pues tardará menos en acabar. Mientras que si eso nos da igual, bien por que no hay muchos ficheros para compilar o porque el ordenador es rápido, le daremos a Build All

Si todo va bien y no hay errores, obtendremos una salida como la siguiente:



Si no, la pantalla será



y al presionar OK saldrá una ventana con un listado de errores encontrados. Si se clika dos veces sobre cada uno de los errores, el programa abrirá el archivo que contiene el error y pondrá el cursor allí donde el ensamblador cree que se produjo (lo cual no tiene por qué ser en el punto donde se debe de solucionar el error), señalándolo con una línea verde. Dependiendo del tipo de error, se solucionará modificando la misma línea u otra en cualquier otro sitio (De todas formas, ayuda más de lo que parece).

Limitaciones

La herramienta con la que trabajamos es una versión de evaluación por lo que está limitada en varios aspectos.

- Genera código objeto de hasta 2kB. Si debiera de ser más grande, generaría un error

- El código fuente puede ser como mucho de 16kB. Si fuera más grande, el editor abriría el programa en modo de sólo lectura y no sería posible modificar el fichero fuente.
- Un proyecto puede tener como mucho 5 ficheros fuente.
- No se puede usar la librería de coma flotante.
- No se puede incluir instrucciones en ensamblador en código fuente en C.

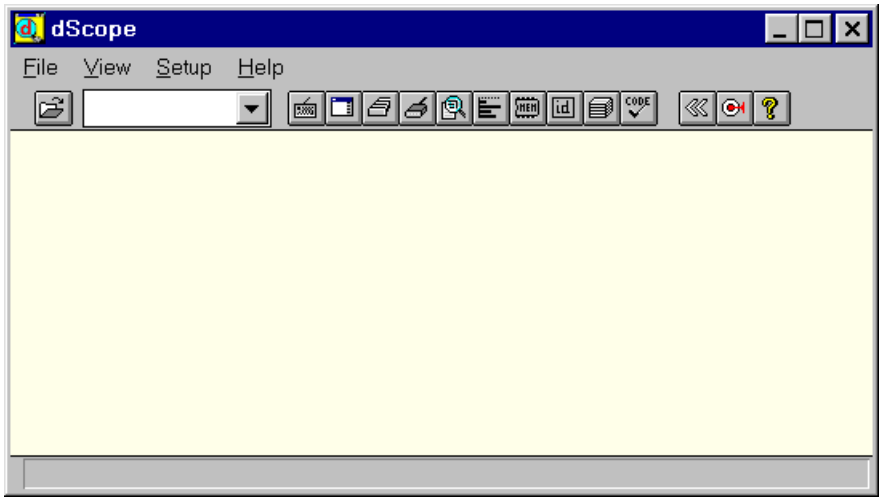
Estas trabas no son impedimento para realizar los programas de prácticas.

Ejecutar el depurador

Una vez compilado con éxito el programa hay que comprobarlo son el simulador. Para ello sólo hay que pulsar el botón  para que éste se ejecute. Cada vez que se pulse el botón se arrancará un nuevo simulador. Las acciones a tomar cuando se arranque el simulador se pueden automatizar tal y como se explica en el apartado Personalización

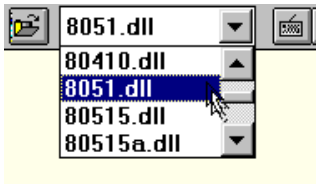
Depurador dScope

El depurador-simulador de µVision, dScope, sirve para comprobar el correcto funcionamiento de nuestro programa antes de llevarlo al diseño final, donde su depuración será más difícil. Desgraciadamente, que funcione en el depurador, no asegura en absoluto que lo vaya a hacer en el hardware final, pues allí existen una multitud de factores que se han sobrentendido en el simulador, que no tienen por que ser así. Por ejemplo, el timing de los circuitos es distinto del pensado, los circuitos externos no funcionan como esperábamos, o cualquier nimiedad por el estilo en la que no habíamos reparado y que SIEMPRE aparece en un diseño de mediana complejidad.



Elegir el microcontrolador

Lo primero que tenemos que hacer es elegir el microprocesador de la familia del 51 con el que vamos a trabajar. En nuestro caso será el 8051 tal y como se ve en la figura.



Nótese que una nueva opción llamada Peripherals se ha cargado en el menú de la ventana. Allí encontramos todos los periféricos de que está dotado nuestro micro.

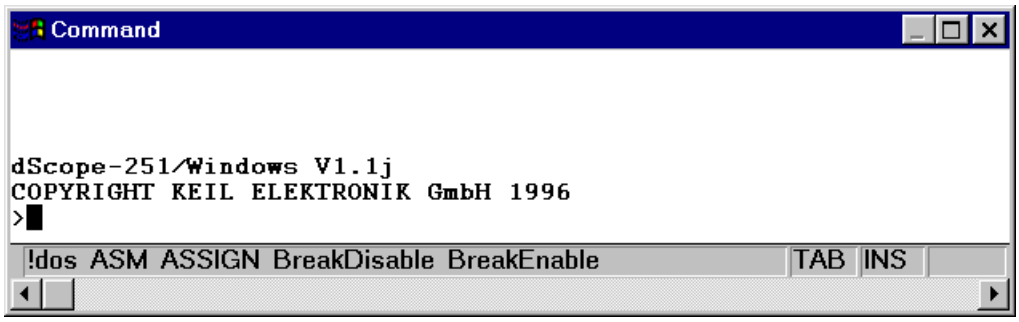
Cargar el programa

En segundo lugar debemos de cargar el programa que queramos simular. El estar trabajando en µVision con un programa no hace que éste se cargue automáticamente al abrir el depurador. Tendremos que pulsar en el sobre que está a la izquierda del archivo dll escogido y cargar el programa. Será el archivo que tenga el nombre del proyecto y que no tiene extensión. Si la ventana de código está abierta se verá el código fuente de nuestro programa. Si no deberemos de pulsar en  para abrirla.

Las ventanas

Las posibilidades que nos ofrece el debugger son varias y en este apartado vamos a comentar cada una de ellas a partir de las ventanas que se pueden visualizar.

 Consola de comandos. Abre una ventana en la que se pueden introducir comandos para la visualización de variables, establecimiento de puntos de ruptura, información del sistema, etc. Algunos de los comandos más útiles se explicarán en la sección Personalización.

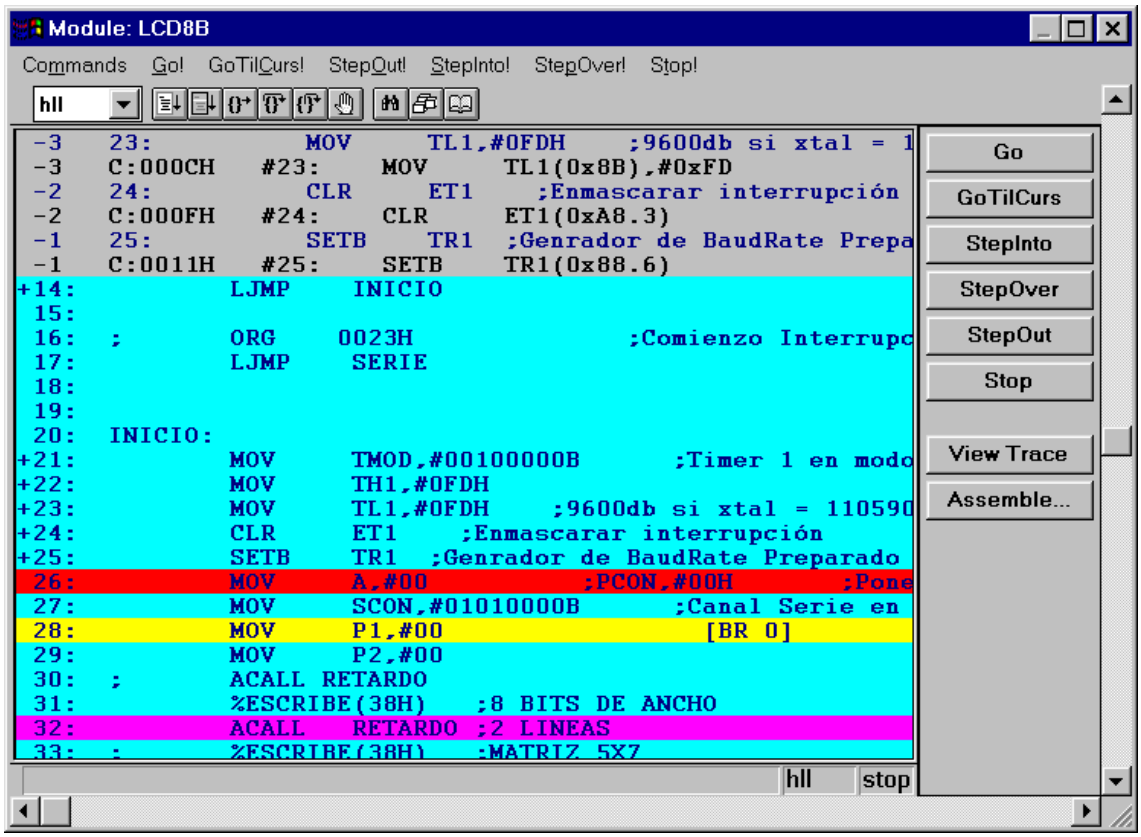


 Ventana de programa. En esta ventana se puede ver el código fuente que se está ejecutando. Se puede visualizar de tres formas.

- HLL: High Level Language. Se ve el código fuente (ensamblador o C) siempre y cuando se hayan establecido los modificadores adecuados para incluir la información para el depurador (Ver la sección Fijar Switches). Es la forma más cómoda de trabajar
- ASM: Assambler. Se visualiza el código ensamblador generado por el compilador.
- MIXED: Mezclado. Con cada línea escrita en lenguaje de alto nivel, le corresponden una o varias en ensamblador. En esta forma se pueden ver las correspondencias (y posibles fallos) de la traducción de C del compilador a lenguaje máquina.

A veces el programa no pasa de un modo a otro correctamente, por lo que lo mejor es no tocar en exceso esas vistas.

La siguiente instrucción que se ejecutará se marca seleccionándola en rojo. La línea donde está el cursor en esos momentos queda marcada en morado, y la línea donde hay un punto de ruptura estará en amarillo. Para poner un punto de ruptura, se clika dos veces con el botón izquierdo al comienzo de la línea en la que se quiere poner el punto de ruptura



Go. Ejecuta el programa en el simulador y este no parará hasta que se encuentre un punto de ruptura (breakpoint) o se pulse el botón de stop.

Go til Curs. Se ejecuta el programa que parará automáticamente cuando llegue a la posición del cursor (o se encuentre un punto de ruptura).

Step Into. Se ejecuta una instrucción. Si la instrucción es una llamada a una subritina, se irá a la primera intrucción de la subrutina. Si el lenguaje es C y el modo es HLL, se ejecutará la sentencia en C, que puede equivaler a varias sentencias de ensamblador. Si la ventana está en modo ASM, aunque el código fuente esté en C, se ejecutará una única instrucción en ensamblador.

Step Over. Ejecuta una instrucción. Pero a diferencia del caso anterior, si ésta es una llamada a una función, esta será ejecutada entera y el programa se parará cuando acabe la rutina.

Step Out. Ejecuta el programa sin parar hasta llegar al punto de regreso de la función (RET o return).

Stop. Detiene la ejecución de programa cuando se está ejecutando en modo Go, o no se llega a ningún punto de ruptura

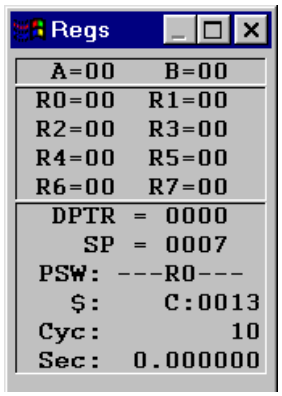
 View Trace Recodrs. Si en la opción de menú Commands de la ventana hemos elegido la opción Trace Records, y hemos ejecutado una serie de instrucciones después, podremos ver cuáles han sido esas instrucciones con esta opción, y además, si vamos seleccionando cada una de las instrucciones que se han ejecutado (que aparecen en fondo gris) podremos ver en la ventana de registros los valores que estos tomaron para esa instrucción. Es como ver la historia pasada del programa.

 Select new module source. Cuando el proyecto consta de varios ficheros fuente (cada uno de ellos es un módulo) podemos ir directamente al que nos interese con esta opción.

 Search for string. Busca una cadena en el código fuente.

Assamble. Se puede cambiar una o varias instrucciones del código del programa si no se tiene el fichero fuente...

 Ventana de registros. Se pueden visualizar directamente los contenidos de los registros A, B y del banco activo, además del DPTR, el SP y el PSW. En el caso de la figura con R0 en el PSW se indica que el banco activo es el cero. El $ indica la instrucción en curso (el PC); Cyc indica el número de ciclos de instrucción ejecutados, y Sec el número de segundos que llevan transcurridos desde el comienzo del programa. Dependerá del valor del cristal con el que se esté trabajando. Véase sección Preferencias para poder cambiarlo. Por defecto se trabaja con un reloj de 12 MHz.

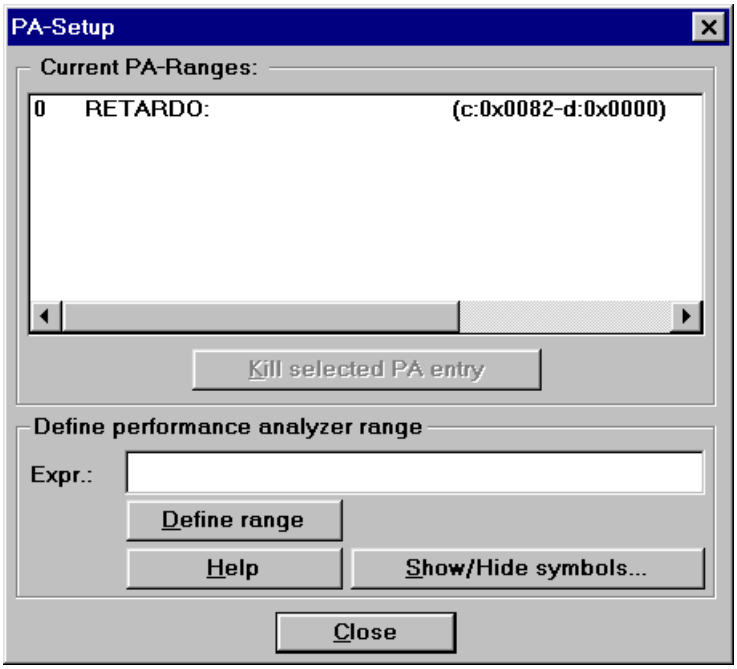


 Serial I/O. En esta ventana se puede ver lo que el micro manda por su puerto serie mientras se ejecuta la aplicación. Se puede ver en ASCII o en hexadecimal. Para simular el efecto de enviar un carácter al puerto serie existen dos formas. Si el programa se está ejecutando, cualquier carácter introducido en la ventana del puerto serie será recibido por el micro. En parado, se puede escribir en la ventana de comando `sin = carácter`, donde carácter es aquel valor ASCII que se quiere que el micro reciba vía serie.

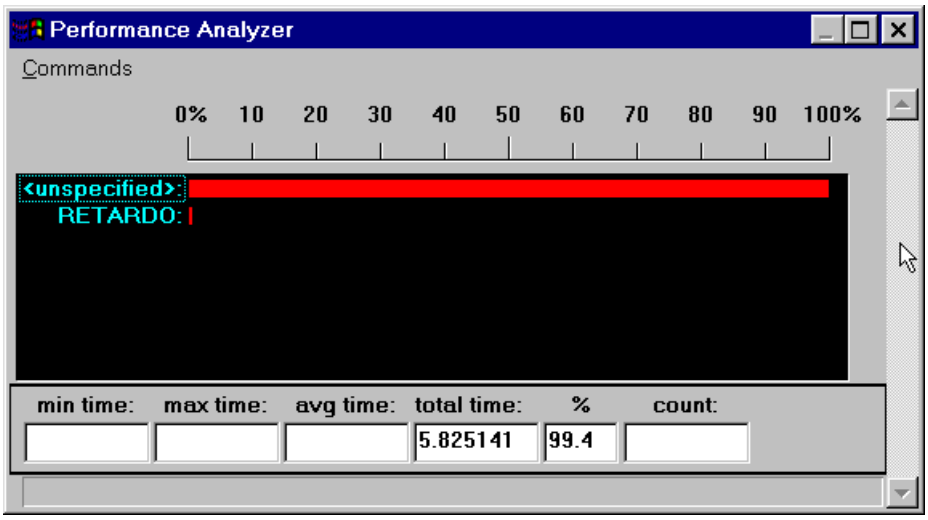

Watch. Ventana donde se pueden visualizar los valores de variables que definamos. Para poner una variable en esta ventana, desde la línea de comando se escribe ws variable, donde variable es el nombre de lo que queremos ver.

Cuando el valor de alguna de las variables que están en la ventana cambia, se resalta en rojo. Para quitar un watch, se escribe en la ventana de comandos `wk número`, donde número es el número que aparece entre paréntesis a la izquierda del nombre de la variable


Performance Analyzer. Establece una comparativa de los tiempos de ejecución de las diferentes funciones que se quieran controlar. Para 'registrar' una función en el analizador hay que darla de alta en Setup->Setup Performance Analyzer.

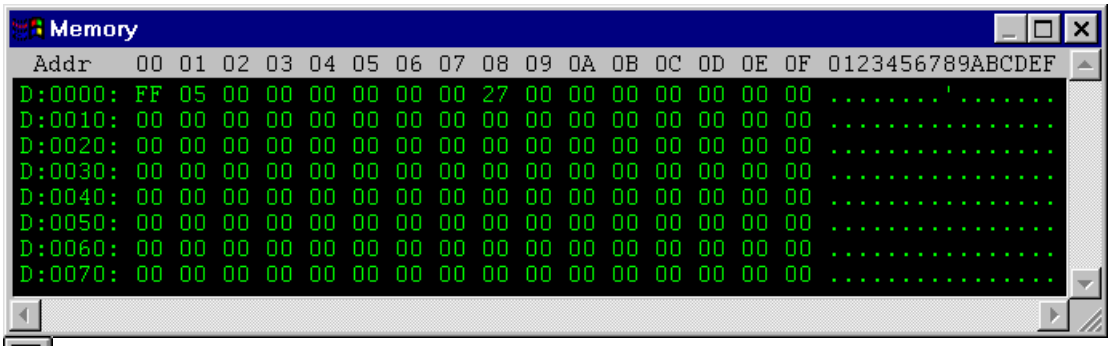


Poniendo en 'Expr.' el nombre de la función cuyo tiempo queremos observar, y pulsando Define Range para incorporarlo a la lista. Desde la línea de comando se puede escribir `PA NombreDeFuncion` y se agregará automáticamente a la lista de funciones a monitorizar.

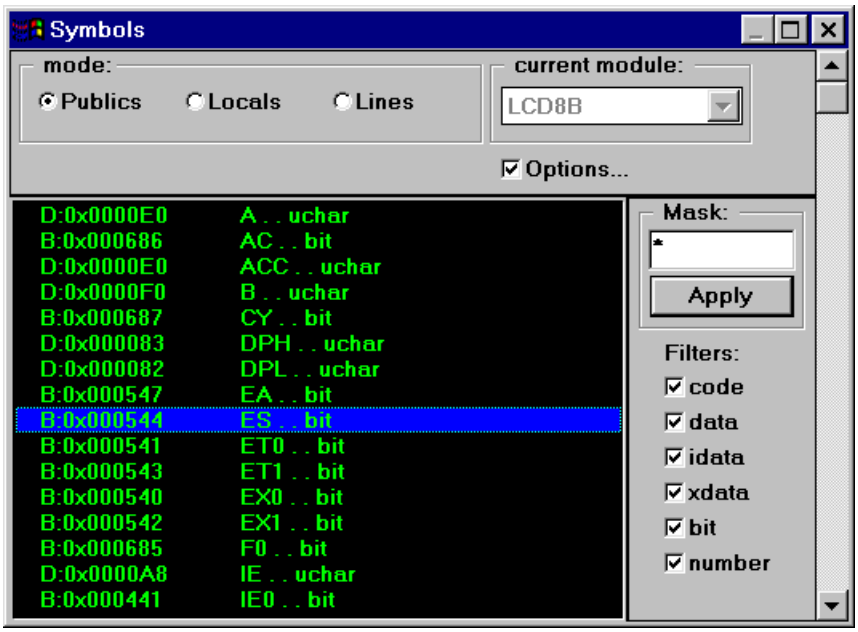


El resultado se observa en un diagrama de tiempos como el que aprece en la figura superiror, donde retardo ocupa un tiempo muy pequeño con respecto al resto del programa.

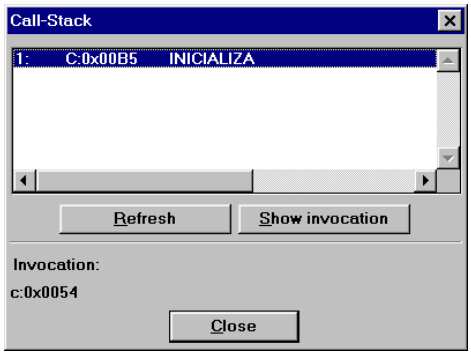
Memory. Se visualiza la zona de memoria elegida. En este caso vemos en la figura el área de memoria interna. Para visualizar la memoria externa hay que escribir en la línea de comando lo siguiente: `D X:DireccionInicio,DireccionFinal` y veremos que la letra de con que comienza cada línea de la ventana es una X. Para volver a visualizar la memoria interna se puede escribir en la ventana de comando bien `D D:DireccionInicio, DireccionFinal` o bien con `D I:DireccionComienzo, DireccionFinal`. Para ver el área de datos direccionables bit a bit se escribe `D b:DireccionComienzo,DireccionFinal`



Symbols. En esta ventana aparecen todos los símbolos del programa, desde los registros del 8051, hasta los definidos por el usuario. Se pueden filtrar para que aparezcan únicamente los relativos al código, o zonas direccionables bit a bit... pulsando en Options. Desde esta ventana se puede arrastrar y soltar cualquier símbolo en el apartado Expr. de las ventanas de Setup->Breakpoints, Setup->Watchpoints y Setup->Setup Performance Analyzer, evitando errores al escribir los símbolos.

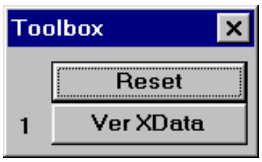


Call Stack. En esta ventana se puede comprobar cual es la profundidad del stack, o dicho de otra forma, cuántas llamadas a funciones se han encadenado hasta ese momento.



Code Coverage. En esta ventana se ofrece información sobre la ejecución o no ejecución de determinadas zonas de un programa. Podemos tener zonas 'muertas' por las que nunca se pasa. Con esta herramienta podemos descubrirlas con cierta facilidad.

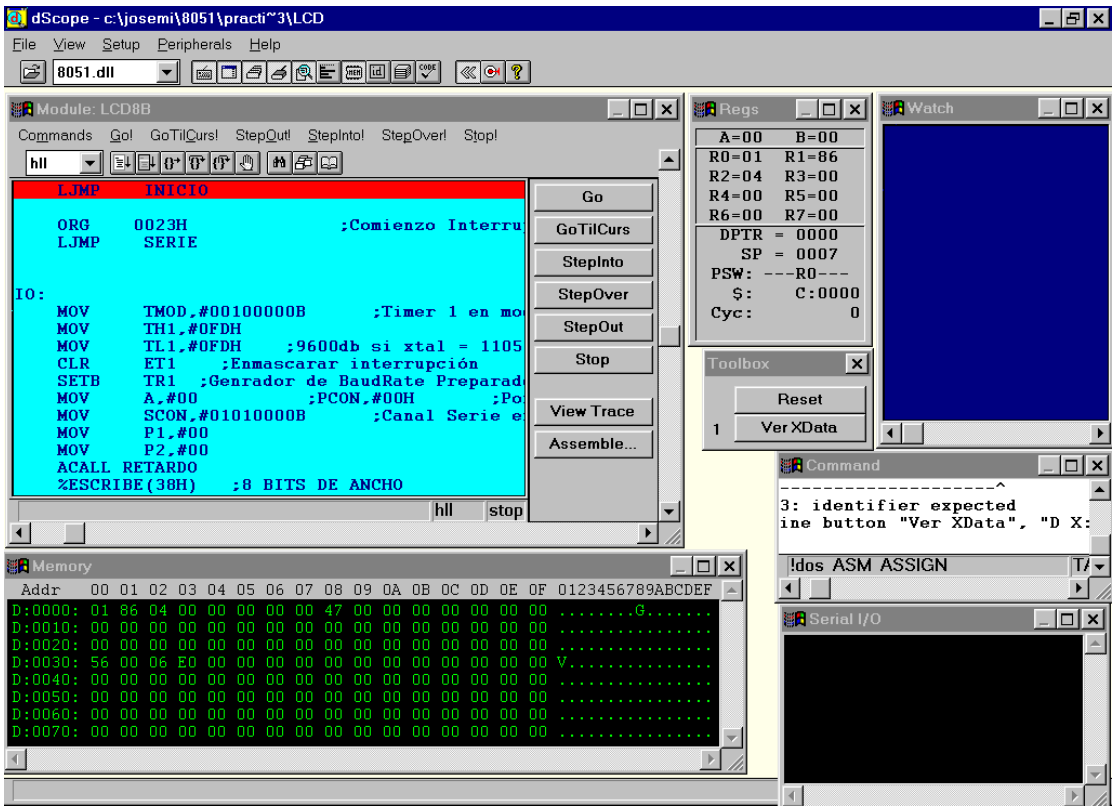
Toolbox. Abre una ventana en la que en principio solo aparece el botón del reset. De forma sencilla se pueden añadir botones que hagan ciertas funciones. Por ejemplo, para cambiar la visualización de la ventana de memoria se puede escribir lo siguiente en la línea de `comando define button "Ver Xdata","D X:0000,1000H"`. Esto añade un botón con la inscripción ver XData y que realiza la función D X:0000,1000H



 Reset. Resetea el programa

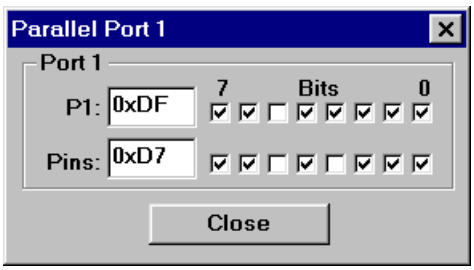
 About. Saca la ventana de información de la versión del producto

Con todas las herramientas que se han comentado se puede disponer de la pantalla de forma que se vean las más usadas. La forma en que se dispongan las ventanas será recordada en la siguiente sesión.

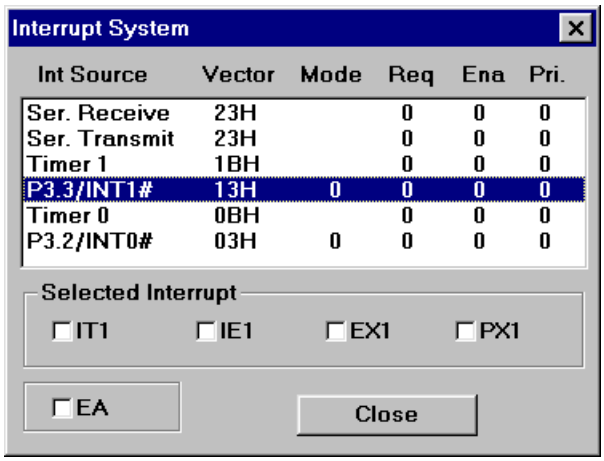


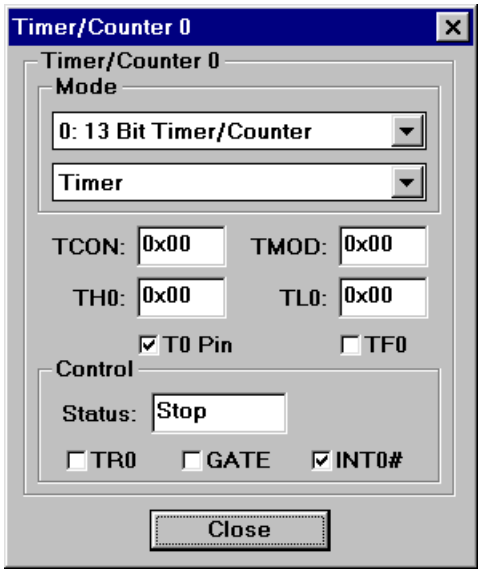
Una vez elegida la DLL con el modelo del microcontrolador que usamos, aparece en el menú la opción Peripherals con todos los periféricos.

- Puertos: Se presentan el valor del registro del puerto y el del pin físico del puerto. El aspa indica un uno. Se puede comprobar como si en el registro hay un cero (no es una entrada), y se intenta cambiar el valor del pin, nos da un error, ya que no leeremos bien el valor lógico.



El resto de las ventanas presentan todos los bits relacionados con cada uno de los periféricos. En cualquier momento se puede modificar el modo de funcionamiento del puerto serie o de un temporizador, simplemente cambiando los valores de los registros en las ventanas correspondientes, caso de que estos no sean los debidos. Lo que no se habrá que olvidar es actualizar el fichero fuente para que refleje los cambios realizados en tiempo de ejecución, caso de que estos posibiliten el correcto funcionamiento del programa.





Personalización y funciones avanzadas.

Existen varias características del dScope que permiten un uso mucho más sencillo del debugger. Crear botones, abrir ventanas para pedir datos concretos, cargar la DLL y el programa de forma automática... Este apartado intenta explicar brevemente alguna de ellas. Para cualquier duda se debe consultar la ayuda del dScope.

Carga automática de la DLL y el programa

Una de las opciones de µVision es la de ejecutar un archivo de comandos previo a la carga de dScope. En Options->dScape Debugger de µVision podemos introducir el nombre de un fichero con la extensión .INI que será el que contenga una serie de comandos que se ejecutarán al entrar en dScope.
Para cargar automáticamente la DLL del 8051 y el programa se deberá escribir lo siguiente en un archivo y salvarlo con la extensión .INI. Después habrá que poner en Options->dScope Debugger del µVision el nombre y path del archivo recién creado.

```
Load 8051.dll
Load prg
```

Donde prg es el nombre del proyecto sin extensión.

Ejecutar hasta cierto punto.

El comienzo del programa, en la dirección 0000H, tiene los vectores de interrupción, que no son demasiado interesantes. Muchas veces lo que queremos es que se ejecute el programa hasta donde realmente empieza, la etiqueta INICIO: en ensamblador o main en C. Para ello añadiríamos una línea al .INI que creamos en el apartado anterior.

```
Load 8051.dll
Load prg    /*Esto es un comentario en el INI*/
g, INICIO   /*o g,main */
```

Pedir la información

Escribiendo `DIR VTREG` en la ventana de comando se devuelve el valor de una serie de parámteros del sistema, como son el valor de los puertos, P0 a P3, la entrada y salida serie, sin y sout, y el valor del cristal. Para cambiar cualquiera de ellos, se debe de escribir en la línea de comando, por ejemlo, `xtal = 11093000`.

Cambiar el valor de un registro

Se escribe en la línea de comando `R3 = 23` o `ACC = 8` o `TMOD = 0x23`

Cambiar el valor de un byte de memoria

Se escribe E Tipo Direccion=Valor. Por ejemplo, E WORD X:0000=0xAA55 Carga en el byte de dirección 0 y en el de dirección uno de memoria externa el valor AA55H. Otros ejemplos son E BIT 0x20.4 = 1, para memoria de bits o E BYTE 0x1A=0xBC

Crear una función

De usuario

Una función de usuario es una rutina que realiza varias tareas y que no acaba hasta que dichas tareas han terminado. Se usan para automatizar diversas acciones como la petición de datos desde una ventana, o cualquier otra cosa que se pueda pensar. No intervienen o tienen relación con eventos que puedan afectar a la ejecución del programa. Se definen de la siguiente forma

```
FUNC TipoRetorno NombreFuncion(Lista, De, Parametros) {
/* Sentencias */
}
```

Donde

- TipoRetorno es uno de los habituales en C
- NombreFunción es el identificador de la función
- Los parámetros (opcionales) son los valores que se pasan a la función para que realicen tareas.
- La llave abierta '{' debe estar en la primera línea.

Un ejemplo de función que abra dos cuadros de diálogo para introducir dos valores es

```
FUNC void Xinput (void)  {
      FinCarrera = getint("Valores de inicio para Final de Carrera");
      Pulsadores = getint("Valores de inicio para los Pulsadores");
      exec("E CHAR x:0xB001 = FinCarrera");
      exec("E CHAR x:0xC000 = Pulsadores");
}
```

Para usar esta función antes habría que haber declarado las variables `FinCarrera` y `Pulsadores` de la siguiente forma

```
exec("define      byte  Pulsadores");
exec("define      byte  FinCarrera");
```

De señal

Las funciones de señales están asociadas a la ejecución del programa. Se pueden generar secuencias de pulsos que imiten la forma en que esos estímulos se producirán en la realidad. La forma de definir una señal SIGNAL es

```
SIGNAL void NombreFuncion(Lista, De, Parametros) {
/* Sentencias */
}
```

Será obligatorio que la función SIGNAL haga una llamada al menos a la función `twatch(NumCiclos).` Esta función detiene la ejecución de la función hasta que pasen twatch ciclos en que se volverá a retomar, en el punto en que se dejó. El siguiente ejemplo imita el comportamiento de un pulsador al ser presionado y soltado.

```
signal void pushbutton (void)  {
      PORT1 |= 0x20;          /* fija Port1.5   */
      twatch (50000);         /* espera 50 ms    */
      PORT1 &= ~0x20;         /* limpia Port1.5 */
}
```

Incluir funciones de otros archivos

Para un diseño más flexible de las funciones se pueden poner en diferentes archivos que son después incluidos en el .INI final Por ejemplo, podemos escribir una función que declare las variables del ejemplo de las funciones de usuario, y meterla en un archivo; después ponemos la función de petición de valores dentro de otro archivo quedando de la siguiente forma.

Archivo Declara.fcn

```
FUNC void Declara(void){
      exec("define      byte  Pulsadores");
      exec("define      byte  FinCarrera");
}
```

Archivo Xinput.fcn

```
FUNC void Xinput (void)  {
      FinCarrera = getint("Valores de inicio para Final de Carrera");
      Pulsadores = getint("Valores de inicio para los Pulsadores");
      exec("E CHAR x:0xB001 = FinCarrera");
      exec("E CHAR x:0xC000 = Pulsadores");
}
```

Archivo .INI

```
load 8051.dll           /* cargar el driver del micro */
load grua               /* cargar el programa */
g,main                  /* ir hasta el main */

define button "Ver XData","D X:0,0xFFFF";/* display memoria externa */
define button "Ver Data", "D d:0";       /* display memoria interna */
define button "Ver BData", "D b:0";      /* display memoria de bits */

include declara.fcn;    /* Primero se hacen las declaraciones */
                  /* de las variables que se van a usar */
declara();        /* se hacen efectivas esas declaraciones */
                  /* si no se llama a la función esas variables */
                  /* todavia no existen*/
include Xinput.fcn;     /* cargar función que usa las variables */
                        /* declaradas */
```

Crear un botón con una función asociada

Para ello definimos un botón de la forma ya explicada y en la parte de la función a realizar, ponemos el nombre de la función a invocar cuando el botón se pulse. Por ejemplo, para incluir la función señal en la botonera del toolbox se pondría.

```
define button "Push Button","pushbutton ()"
```

o bien en el .INI anterior

```
define button "Pedir Datos","Xinput()";/* activar funcion con boton */
```

Existen numerosas opciones más que pueden ser investigadas en la ayuda del dScope.

Anexo 1 Formato Intel Hex

Se trata de un formato normalizado mediante el cual un programa ejecutable se almacena en un fichero como un conjunto de datos ASCII para su posterior transferencia

Dicho fichero se divide en una serie de registros cada uno de los cuales puede contener hasta 256 bytes de código utilizable. Cada registro se organiza en una serie de campos cuyo formato se muestra a continuación.

Marca de reg.	Tamaño de reg.	Direc. de inicio	Tipo de registro	Bytes de datos	Byte checksum
:	##	aaaa	tt	dd....dd	Cc

Donde:

- : Es el carácter de inicio de registro.
- ## Es un valor ASCII hexadecimal de dos dígitos que indica el tamaño del registro. Es decir, el número de bytes de datos que lo compone (máx. 256). Si el registro es del tipo 01, este campo debe valer 00.
- aaaa Cuatro dígitos hexadecimales expresados en ASCII que corresponden con la dirección inicial de memoria donde deben almacenarse, de forma consecutiva, los bytes de datos. Si el registro es del tipo 01, este campo debe valer 0000.
- tt Es un valor ASCII hexadecimal de dos dígitos que representan el tipo de registro.
  - 00 - Registro de datos
  - 01 - Último registro (fin de fichero)
- dd...dd Son dos caracteres ASCII por byte de datos. Habrá tantos bytes por registro como indique el campo ##. Si el tipo de registro es 01 (último registro) no habrá ningún byte de datos.
- cc Son dos caracteres ASCII hexadecimal que representan el byte del checksum. Este byte se obtiene de forma tal que, al sumar todos los bytes de los distintos campos de un registro comenznado desde ## y finalizando con el propio cc, se obtenga siempre el valor 00.

Ejemplo:

Un fichero en formato Intel Hex contiene la siguiente información.

:06100000231A45BEE46264

:060000002300A8A917046B

:00000001FF

1er registro	##	06h
	aaaa	1000h
	tt	00
	dd...dd	23h,1Ah, 45h, BEh, E4h, 62h
	cc	64h (06h+10h+00h+23h+1Ah+45h+Beh+E4h+62h+64h = 00)

2º registro	##	06h
	aaaa	0000h
	tt	00h
	dd...dd	23h, 00h, A8h, A9h, 17h, 04h
	cc	6Bh (06h+00h+00h+23h+00h+A8h+A9h+17h+04h+6Bh = 00)
3er registro	##	00h
	aaaa	0000h
	tt	01h (último registro)
	cc	FFh (00h+00h+00h+01h+FFh = 00)

Bibliografía

- dScope for Windows Help, Archivo de ayuda de dScope dsw51.hlp
- A51 Assambler User's Guide, Keil Software 04.95
- C51 Compiler User's Guide, Keil Software 01.97
- 8051 Utilities, Keil Software
- dScope for Windows User's Guide Keil software 01.97