Práctica 4
Introducción a la Programación en C de CCS de los Microcontroladores PIC

- Introducción y objetivos
- Características propias del lenguaje C del compilador CCS
  - Tipos de datos
  - Constantes
  - Directivas básicas
  - Funciones
- El compilador C de CCS
  - Características básicas
  - Integración en el IDE MPLAB
- Tareas a realizar

Introducción

- Objetivos:
  - Apreciar las ventajas e inconvenientes de gestionar el funcionamiento de los microcontroladores mediante lenguajes de medio o alto nivel.
  - Conocer las características básicas del lenguaje C del compilador de CCS que suponen añadidos respecto al C estándar.
  - Conocer las principales características del compilador C de CCS y su integración en el IDE MPLAB.

Introducción

- Como es sabido, los lenguajes de medio o alto nivel, se basan en una nomenclatura y sintaxis más cercanas a las operaciones lógico-matemáticas que el lenguaje ensamblador, con lo que, en general, se facilita la labor del programador y se producen códigos fuente más compactos.
- Por contra, el código maquina generado a partir de lenguajes de medio o alto nivel puede no ser tan compacto como el generado a partir del código en ensamblador.
- En cualquier caso, la gestión mediante C de las funcionalidades de los microcontroladores resulta interesante y atractiva, especialmente en lo referente a controlar determinados aspectos.
- En lo que resta de curso nos centraremos justamente en cómo gestionar varios aspectos de la MCU de Microchip PIC16F877A mediante el compilador C de CCS, que es compatible con el entorno de desarrollo integrado MPLAB.

Características propias del lenguaje C del compilador CCS

- El compilador C de CCS se basa en un lenguaje C más o menos estándar, pero añade elementos propios para facilitar la gestión de las funciones de los microcontroladores (principalmente librerías, directivas del preprocesador y funciones).
- Muchos de estos elementos propios sirven para gestionar aspectos muy específicos del funcionamiento del microcontrolador (E/S, interrupciones, etc.) mientras que otros tienen un carácter más genérico.
- Nos centraremos en los elementos “específicos” al tratar la gestión en C de aspectos concretos de los microcontroladores PIC, pero previamente abordaremos los elementos “genéricos” más básicos.
- En cualquier caso, a lo largo del curso no veremos todos y cada uno de los elementos propios del lenguaje C de CCS. Sin embargo , el manual completo de C de CCS está disponible en Moodle, de modo que la información completa sobre esta versión de C está accesible.
- IMPORTANTE: se asume un conocimiento previo del C estándar, que NO repasaremos.

Tipos de datos del compilador C de CCS:

Tipo	Tamaño	Rango	Descripción
Int1 **Short**	1 bit	0 a 1	Entero de 1 bit
Int **Int8**	8 bits	0 a 255	Entero de 8 bits
Int16 **Long**	16 bits	0 a 65535	Entero de 16 bits
Int32 **Long Long**	32 bits	0 a 2^{32}-1	Entero de 32 bits
Float	32 bits	$\pm 1.175 \times 10^{-38}$ a $\pm 3.402 \times 10^{38}$	Punto flotante
Char	8 bits	0 a 255	Carácter
Void	-	-	Sin valor
Signed Int8	8 bits	-128 a 127	Entero con signo
Signed Int16	16 bits	-32768 a 32767	Entero largo con signo
Signed Int32	32 bits	$-(2^{31})$ a 2^{31}-1	Entero 32 bits con signo

Tipos de constantes del compilador C de CCS:

Tipo	Ejemplo
Decimal	123
Octal	0123
Hexadecimal	0x123
Binario	0b010010
Carácter	'x'
Carácter Octal	'\010'
Carácter Hexadecimal	'\xA5'
Cadena (String)	“abcdef”
Carácter especial	\n Line Feed - Same as \x0a \r Return Feed - Same as \x0d \t TAB - Same as \x09 \b Backspace - Same as \x08 \f Form Feed - Same as x0c \a Bell - Same as \x07 \v Vertical Space - Same as \x0b \? Question Mark - Same as \x3f \' Single Quote - Same as \x22 \" Double Quote - Same as \x22 \\ A Single Backslash - Same as \x5c

Directivas básicas propias del compilador C de CCS

- El compilador de C de CCS incorpora varias directivas del preprocesador propias, además de las propias del C estándar.
- A su vez, algunas directivas habilitan el uso de determinadas funciones que el C de CCS incorpora pero que no son propias del C estándar.
- A continuación nos centraremos en las directivas propias “genéricas” más básicas del C de CCS, principalmente en las relativas a especificación y configuración de dispositivos y control de memoria.

Directivas de especificación y configuración de la MCU

- **`#DEVICE chip tipo_puntero .`** Indica al compilador el modelo concreto (**`chip`**) de MCU, lo que obviamente determina la RAM, la ROM y el juego de instrucciones. En modelos con más de 256 bytes de RAM, opcionalmente se puede seleccionar entre punteros de 8 o 16 bits mediante el valor de **`tipo_puntero`**. (**`*=8`** ó **`*=16`**). Ejemplos:

```
#DEVICE PIC16F877A
#DEVICE PIC16F877A *=16
#DEVICE *=16 //solo después de declarar el chip
```

- **`#INCLUDE <chip_number.h> .`** Incluye un fichero de cabecera donde se indica al compilador el modelo concreto de microprocesador (es decir, el fichero contiene un **`#DEVICE`**), y donde además se definen varias constantes útiles (como los pines). Ejemplo:

```
#INCLUDE <16F877A.h> //el número no lleva ´´PIC´´ delante
```

Directivas de especificación y configuración de la MCU

- **`#FUSES options .`** Establece los valores de la palabra de configuración de la MCU mediante la serie de opciones indicadas en **`options`**. Ejemplos:

```
#FUSES HS, NOWDT, NOBROWNOUT
#FUSES XT, WDT, BROWNOUT
```

La palabra de configuración también puede establecerse directamente mediante la asignación de un valor hexadecimal. Ejemplo:

```
#FUSES 1=0x0602 //El 1 indica palabra de configuración 1
```

La configuración establecida con **`#FUSES`** realmente no afecta a la compilación en C, pero se incluye en los ficheros de salida.

Las opciones posibles para **`#FUSES`** dependen de la MCU concreta. En la transparencia siguiente pueden verse todas las opciones del PIC16F877A .

Directivas de especificación y configuración de la MCU

- Posibles opciones de **`#FUSES`** para el PIC16F877A:

LP	Low power osc < 200 khz
XT	Crystal osc <= 4mhz for PCM/PCH , 3mhz to 10 mhz for PCD
HS	High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
RC	Resistor/Capacitor Osc with CLKOUT
NOWDT	No Watch Dog Timer
WDT	Watch Dog Timer
NOPUT	No Power Up Timer
PUT	Power Up Timer
PROTECT	Code protected from reads
NOPROTECT	Code not protected from reading
NOBROWNOUT	No brownout reset
BROWNOUT	Reset when brownout detected
NOLVP	No low voltage programming, B3(PIC16) or B5(PIC18) used for I/O
LVP	Low Voltage Programming on B3(PIC16) or B5(PIC18)
NOCPD	No EE protection
CPD	Data EEPROM Code Protected
WRT_5%	Lower 255 bytes of Program Memory is Write Protected
WRT_25%	Lower quarter of Program Memory is Write Protected
WRT_50%	Lower half of Program Memory is Write Protected
NOWRT	Program memory not write protected
NODEBUG	No Debug mode for ICD
DEBUG	Debug mode for use with ICD

Directivas de especificación y configuración de la MCU

- **`#USE DELAY (CLOCK=valor_frecuencia)`**. Establece la frecuencia del procesador, mediante un valor (**`valor_frecuencia`**), que puede expresarse en Hz (ciclos de reloj por segundo) mediante un valor sin sufijo, en KHz (añadiendo al valor el sufijo K o KHZ) o en MHz (añadiendo al valor el sufijo M o MHZ) . Ejemplos:

```
#USE DELAY (CLOCK=8000000)
#USE DELAY (CLOCK=8000K)
#USE DELAY (CLOCK=8M)
```

Esta directiva puede emplearse con "**OSC**", "**XTAL**", "**INT**" ó "**RC**" en lugar de (o además de) "**CLOCK**" para indicar el tipo de reloj, en cuyo caso se modifica el tipo de reloj que pudiera haberse indicado con **#FUSES**. Ejemplos:

```
#USE DELAY(XTAL=20MHZ)
#USE DELAY (CLOCK=20MHZ, XTAL)
```

Directivas de control de memoria

- **`#BYTE x=y`.** Donde **`x`** es un identificador e **`y`** puede ser otro identificador o una constante. El efecto es crear (o reubicar, si el identificador ya existe) una variable de tipo int8 (byte) con identificador **`x`** en la posición de memoria indicada por **`y`** (si **`y`** es una constante), o en la posición de memoria asignada a **`y`** (si **`y`** es un identificador). Ejemplos:

```
#BYTE portb=0x6
#BYTE trisb=0x86
```

- **`#WORD x=y`.** Similar a **`#BYTE`** pero con variables de tipo int16.

- **`#BIT x=y.z`.** Donde **`x`** es un identificador, **`y`** puede ser otro identificador o una constante, y **`z`** es un valor entre 0 y 7. El efecto es crear una variable de tipo int1 (bit) con identificador **`x`** que corresponde al bit **`z`** en el byte de la memoria indicado por **`y`** (si **`y`** es una constante), o en el byte de la memoria asignado a **`y`** (si **`y`** es un identificador). Ejemplos:

```
#BIT portb0=0x6.0
#BIT trisb7=trisb.7
```

Directivas de control de memoria

- **`#ORG inicio, fin`** . Indica un segmento de la memoria de programa (mediante sus posiciones de inicio y fin) donde si situará el código que se indica a continuación.

- **`#ZERO_RAM .`** Pone a cero todos los registros internos que puedan usarse para mantener variables, antes de la ejecución del programa.

- **`#RESERVE posicion1, posicion2,..`**

  `#RESERVE posicion_inicial:posicion_final`

  Permite reservar posiciones de la RAM, o un rango de posiciones, para uso del compilador. Ejemplos:

```
#RESERVE 0x20, 0x21, 0x22
#RESERVE 0x20:0x22
```

Otras directivas

- **`#ASM, #ENDASM`.** Permiten utilizar código ensamblador dentro de un programa en lenguaje C. Se sitúan respectivamente justo antes y justo después del código ensamblador. Ejemplo:

```
void funcion_en_ensamblador(){
 #asm
  [instruccion_ensamblador_1]
  [instrucción_ensamblador_2]
  .
  .
  [instrucción_ensamblador_ultima]
 #endasm
 return();
}
```

IMPORTANTE: Intentaremos NO usar estas directivas y el código ensamblador dentro de programas en C salvo que sea totalmente inevitable (o sea, en la práctica, no las usaremos nunca)

Actividad (10 minutos)

- Basándonos exclusivamente en algunas de las directivas del C de CCS que hemos visto hasta ahora, y teniendo en cuenta la organización de la memoria del PIC16F877A vista en el tema 3 y en la práctica 2, es posible (por ejemplo) activar uno de los leds conectados al puerto B (por ejemplo, en el pin RB 7) ¿Se te ocurre un programa en C de CCS que tenga este efecto?

Funciones propias del compilador C de CCS

- Como sucede con las directivas, el compilador de C de CCS incorpora varias funciones que no se encuentran en el C estándar.
- A continuación nos centraremos en las funciones propias “genéricas” más básicas del C de CCS, principalmente en las relativas a retardos, operaciones con bits y control del *watchdog timer*.

Funciones de retardo

- **`DELAY_CYCLES(retardo_en_ciclos_instrucción);`** . Introduce el retardo en ciclos de instrucción indicado como parámetro. Es decir, equivale a introducir tantos NOPs como se indique en el parámetro.

- **`DELAY_MS(retardo_en_milisegundos);`** . Introduce el retardo en milisegundos indicado como parámetro. Requiere que se haya declarado antes una directiva **`#USE DELAY (CLOCK=valor_frecuencia)`**.

- **`DELAY_US(retardo_en_microsegundos);`** . Introduce el retardo en microsegundos indicado como parámetro. Requiere que se haya declarado antes una directiva **`#USE DELAY (CLOCK=valor_frecuencia)`**.

Funciones de operaciones con bits

- **`BIT_CLEAR (var, x);`**. Pone a 0 el bit **`x`** (valor de 0 a 7) de la variable **`var`**.
- **`BIT_SET (var, x);`** Pone a 1 el bit **`x`** (valor de 0 a 7) de la variable **`var`**.
- **`BIT_TEST (var, x);`** Devuelve el valor del bit **`x`** de la variable **`var`**.
- **`SWAP (var);`** Los 4 bits de mayor peso y los 4 de menor peso de la variable **`var`** se intercambian.
- **`ROTATE_LEFT(address, bytes)`**. Rota un bit hacia la izquierda en la posición de la memoria indicada por **`address`**. Esta rotación afecta al número de bytes indicado como parámetro en **`bytes`**.
- **`ROTATE_RIGHT(address, bytes)`**.Similar a la anterior, pero con rotación a la derecha.

Funciones de operaciones con bits

- **`SHIFT_LEFT(address, bytes, value).`** Desplaza (sin rotación, eso es, sin retroalimentación) un bit hacia la izquierda en la posición de la memoria indicada por **`address`**. Este desplazamiento afecta al número de bytes indicado como parámetro en **`bytes`**. El bit que pierde su valor en el desplazamiento toma el valor indicado en "**`value`**".

- **`SHIFT_RIGHT(address, bytes, value).`** Similar a la anterior, pero con desplazamiento a la derecha.

Funciones de control del WDT

- **`RESTART_WDT()`** . Reinicia el contador del *watchdog*, en caso de que éste esté habilitado.

- **`SETUP_WDT(mode)`** . Indica el valor del *timeout* del *watchdog* mediante el valor indicado en **`mode`**. Los posibles valores de **`mode`** son: WDT_18MS, WDT_36MS, WDT_72MS, WDT_144MS,WDT_288MS, WDT_576MS, WDT_1152MS, WDT_2304MS.

  Ejemplo:

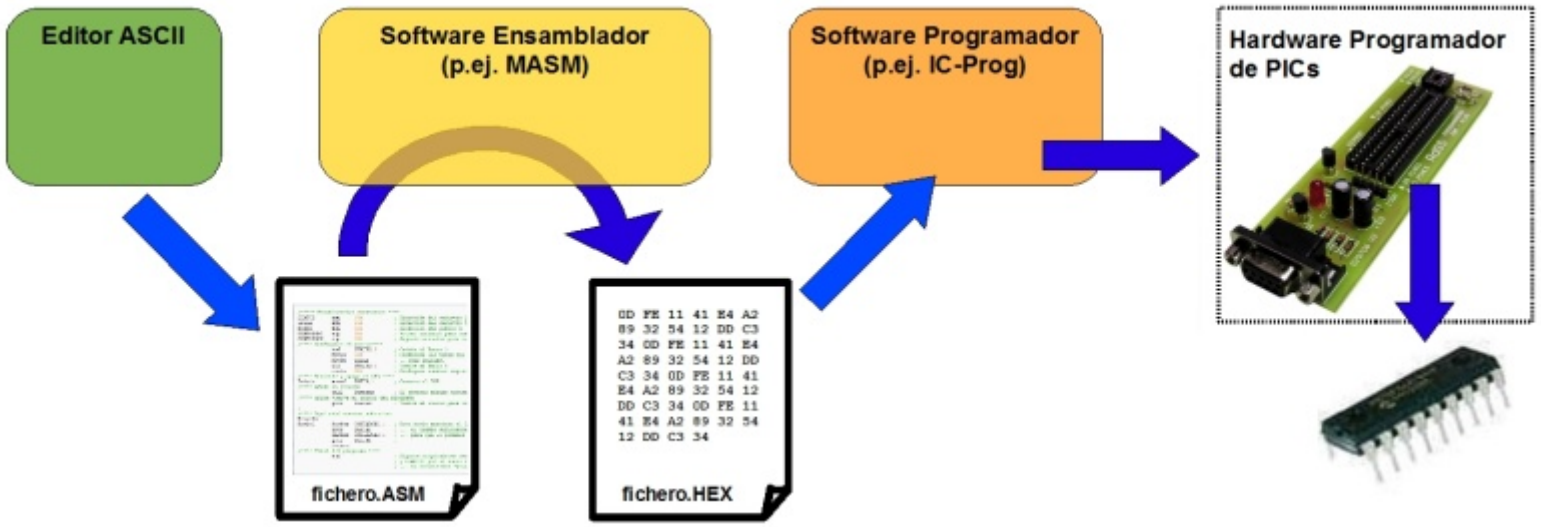
```
#fuses WDT
setup_wdt(WDT_18MS);
void main(){
  while(1){
       restart_wdt();
       operaciones();
  }
}
```

Características básicas del compilador C de CCS

- El compilador C de CCS se basa en un lenguaje C más o menos estándar, y añade elementos propios para facilitar la gestión de las funciones de los microcontroladores de la marca Microchip, principalmente librerías, directivas del preprocesador y funciones.
- Realmente, existen tres versiones del compilador que suelen coexistir dentro de un mismo IDE:
  - PCB: para MCUs de 12 bits .
  - PCM: para MCUs de 14 bits.
  - PCH: para MCUs tipo PIC18 bits.
- Existen IDEs propios de CCS (PCW y PCWH) que permiten trabajar con estos compiladores, pero en nuestro caso trabajaremos con los compiladores integrados en MPLAB.

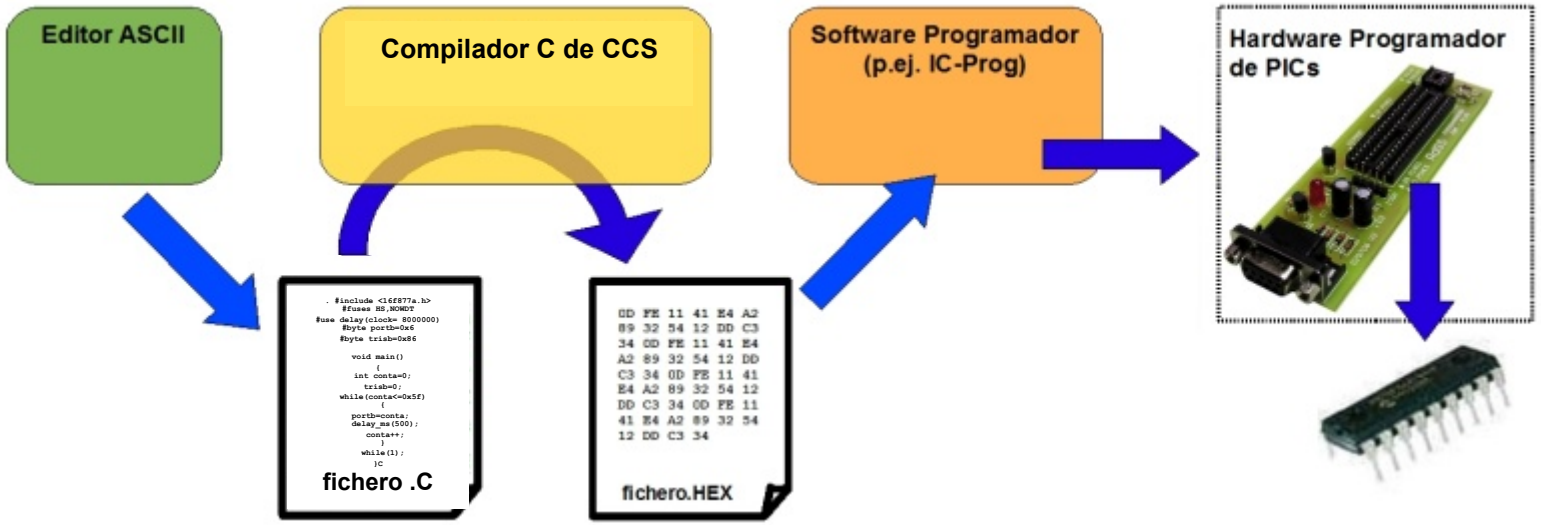
Integración del C de CCS en MPLAB

- La metodología seguida hasta ahora (código ASM) es la siguiente:



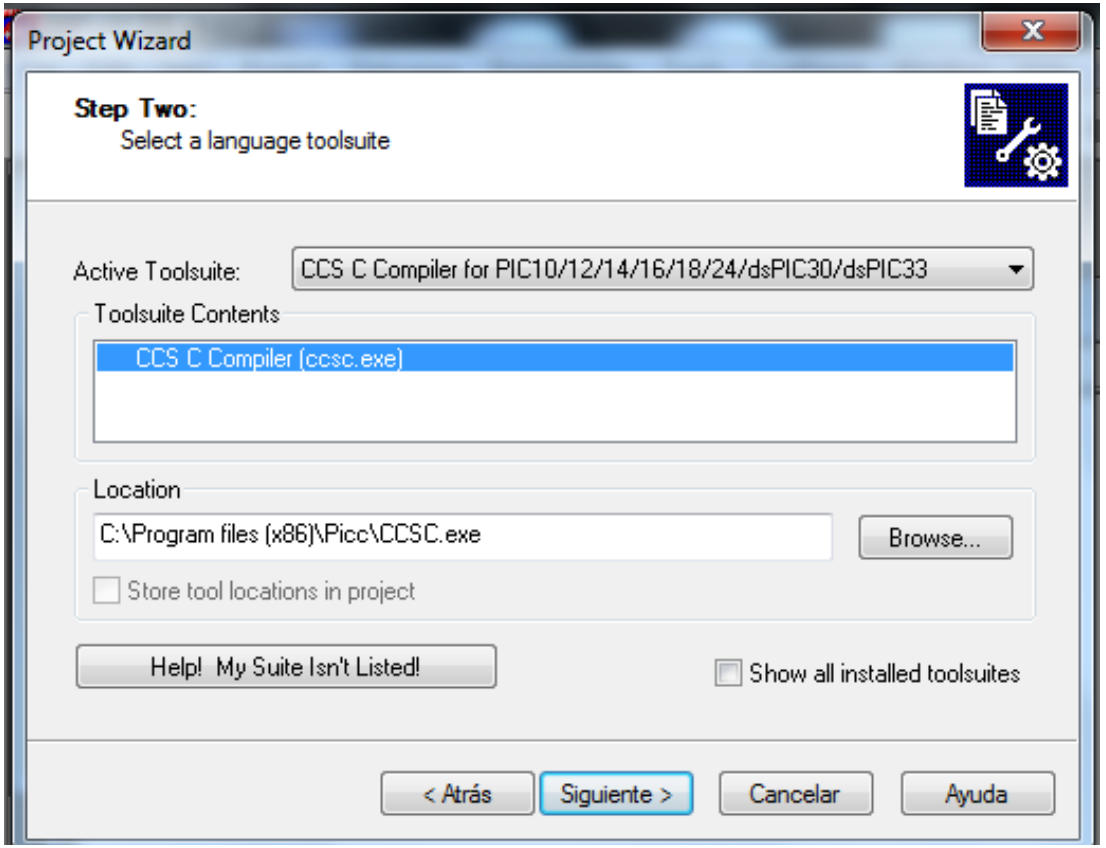
Integración del C de CCS en MPLAB

- A partir de ahora la metodología seguida será la misma pero sustituyendo el software ensamblador por código en C de CCS:



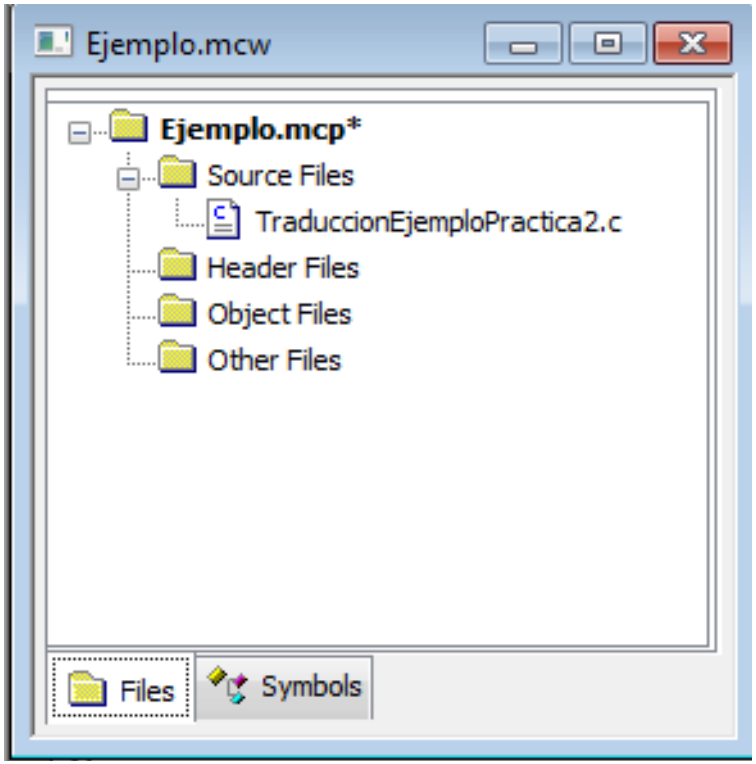
Integración del C de CCS en MPLAB

- Para usar el compilador C de CCS en MPLAB basta con seleccionarlo como lenguaje de programación, por ejemplo en el paso 2 del *"Project Wizard"*:



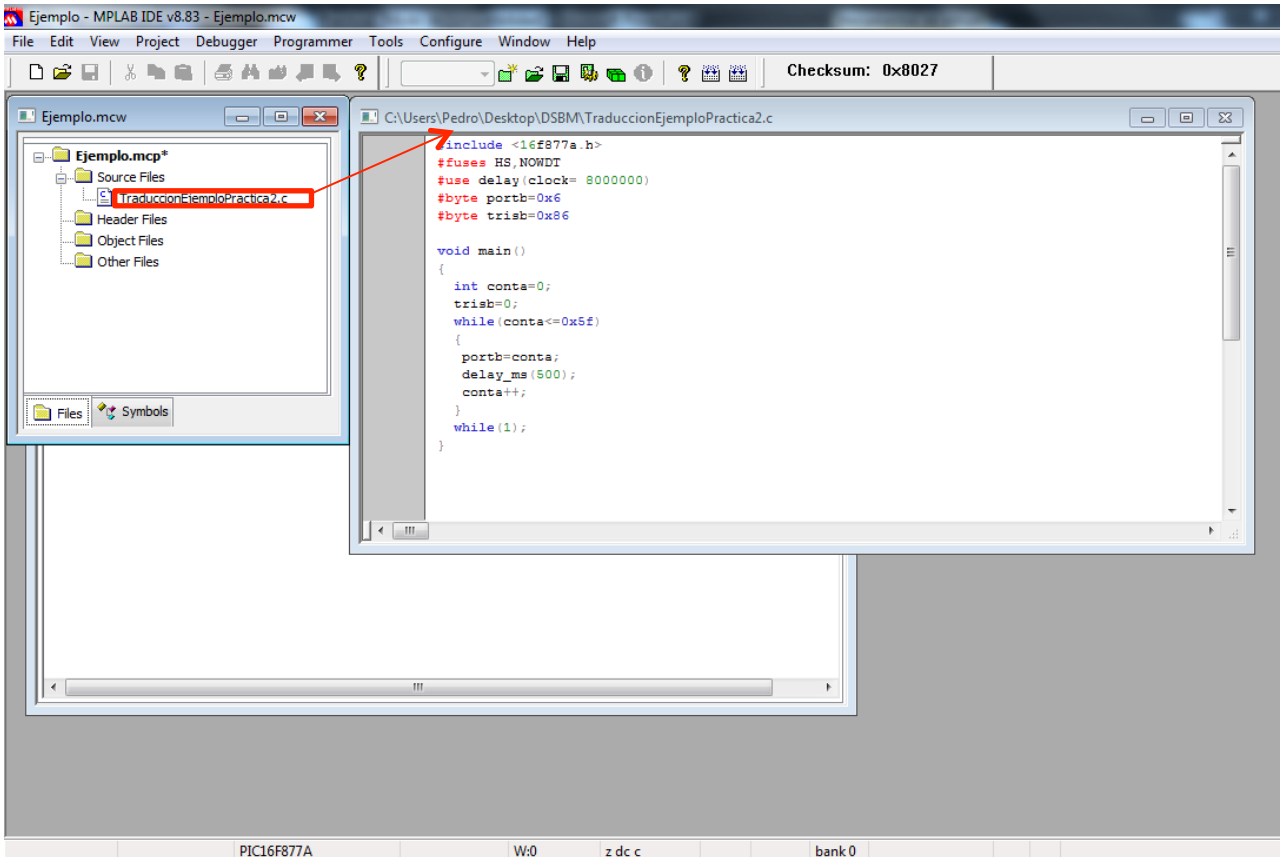
Integración del C de CCS en MPLAB

- Al igual que cuando se usa MPASM, se pueden añadir ficheros al proyecto en el paso 4 del "*Project Wizard*" o posteriormente, una vez creado el proyecto. Obviamente, en este caso los ficheros fuente serán ficheros .C:



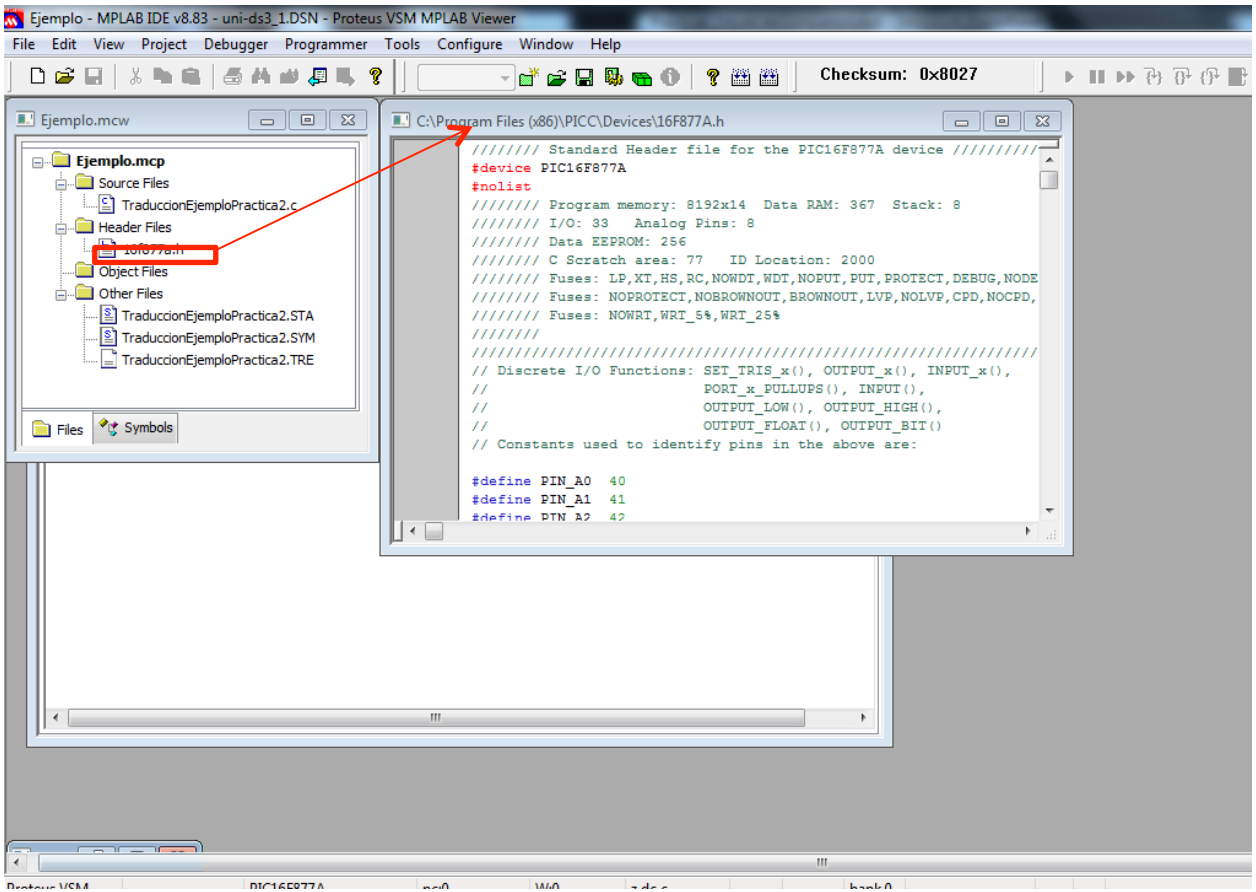
Integración del C de CCS en MPLAB

- Los ficheros fuente .C añadidos son editables desde el propio MPLAB de forma similar a los ficheros fuente . asm:



Integración del C de CCS en MPLAB

- Al compilar el fichero fuente, los ficheros de cabecera referenciados (por ejemplo, los .h de dispositivos) se añaden al proyecto automáticamente:

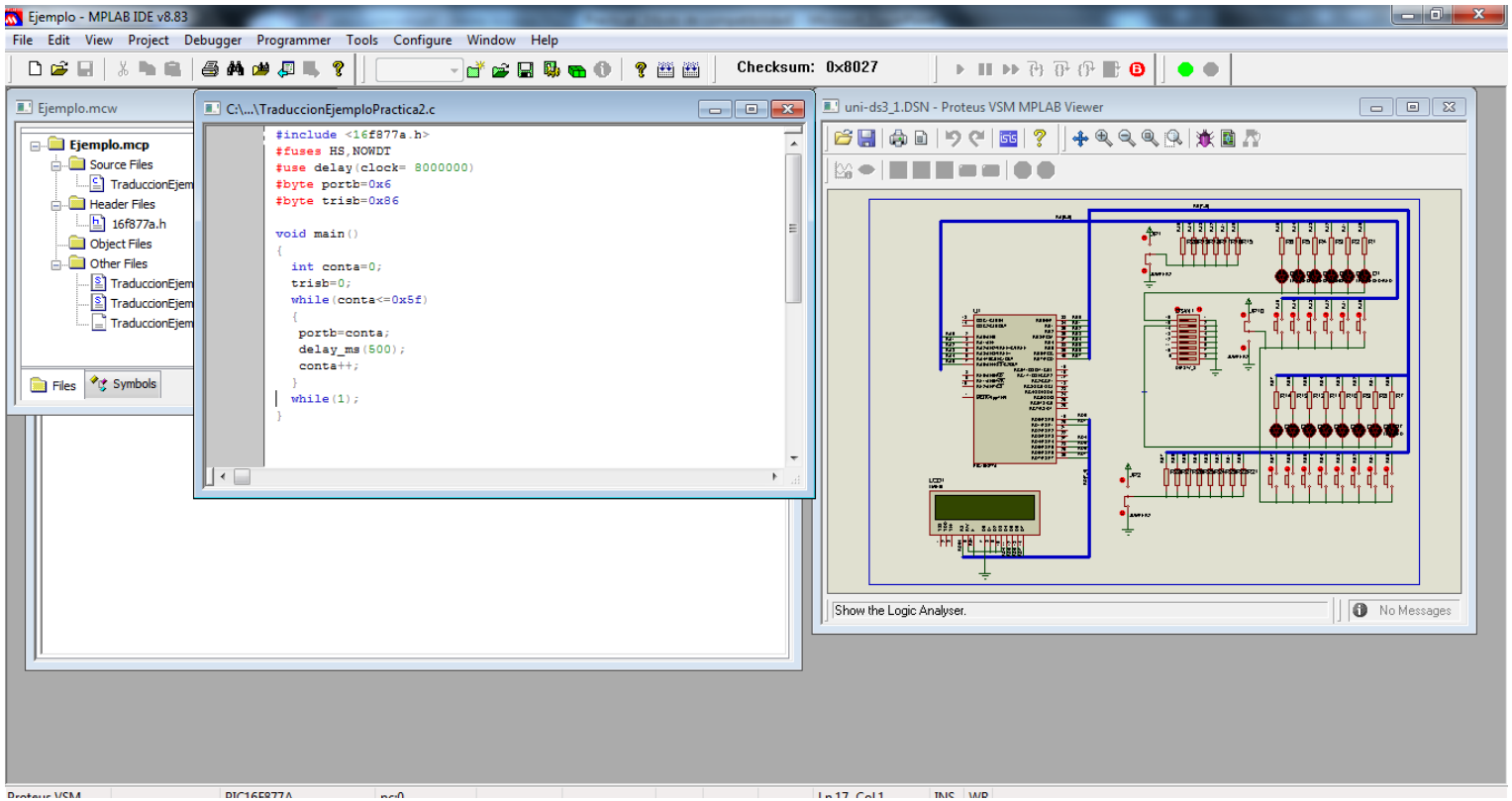


Integración del C de CCS en MPLAB

- Al construir el proyecto se generan varios ficheros de salida. Los más importantes son:
  - .HEX : contiene el código hexadecimal generado para transferirlo a la MCU.
  - .STA : contiene estadísticas sobre el uso de la RAM, la ROM, etc.
  - .SYM : fichero de símbolos con información sobre la locaclización de registros y variables.
  - .TRE: “tree file”, que contiene información sobre las funciones y las llamadas a función.
  - .LST: contiene una traducción del código fuente C a ensamblador. Es posible controlar lo que se vuelca a este fichero con las directivas **`#LIST`** (se lista lo que hay a continuación) y **`#NOLIST`** (no se lista lo que hay a continuación).

Integración del C de CCS en MPLAB

- Finalmente, la compilación puede simularse con un diseño basado en un debugger como ISIS Proteus de la forma habitual, con las opciones deseadas (paso a paso o de una vez, con o sin breakpoints, etc.):



Tareas a realizar

Simular en C de CCS integrado en MPLAB el programa desarrollado en la tarea de la página 15, transfiriéndolo luego a la placa UNI DS3:

Esta práctica no tiene asociada entrega.