Manual de Usuario

Compilador de C para el microcontrolador Microchip PIC 16F877 version 0.9

Autor: Pedro José Ramírez Gutiérrez

Director: David Santo Orcero

7 de abril de 2007

Índice general

Capítulo 1

Manual de usuario

1.1. Introducción

El porting de GCC a PIC16F877 permite al compilador GCC generar código para el microcontrolador PIC16F877. Comenzó como un proyecto fin de carrera a un solo mirocontrolador, pero con la idea clara de sentar las bases que permitan expandirlo a todos los microcontroladores PIC16F de Microchip. Por el momento se soportan a nivel base 3 microcontroladores, pero permite adecuar el código a través de linea de comandos a otros muchos controladores PIC16F.

El porting de GCC a PIC16F877 puede redistribuirse y/o modificarse bajo los términos de la licencia GPL.

Este manual documenta como usar GCC para compilar código en C para PIC16F. También se documentan las herramientas asociadas y las características propias del porting de GCC a PIC.

1.2. Introducción a las herramientas

Para programar los microcontroladores en C con GCC, necesitamos lógicamente el compiador GCC, pero además otras cuantas utilidades. Lo mínimo son el compilador GCC compilado para la arquitectura PIC, las utilidades GPutils y un sistema para programar el PIC fisicamente. Vamos por partes:

1.2.1. PIC-GCC

PIC-GCC es el compilador de C para PIC. Con él podemos transformar un código fuente en C a código en ensamblador.

La sintaxis básica es:

```
pic-gcc [opciones] codigo.c
```

donde codigo.c es el archivo que contiene el programa o código C que deseamos compilar. [opciones] representa a las opciones que nos permite cambiar el modo de operar del compilador. Las principales son:

- -E : Solamente preprocesa, no compila, ensambla o enlaza. Sirve para ver que las macros se extedian como se espera.
- -S : Solamente compila, no ensambla o enlaza. También se preprocesa el código. Con esta opción obtenemos el código en ensamblador.
- -c : Compila y ensambla pero no enlaza. Este es el caso que más usaremos ya que nos da un archivo objeto para la fuente en C. Faltaría enlazarlo para que fuera realmente ejecutable.
- -o <fichero>: coloca la salida en el fichero indicado, por si se desea alterar el nombre por defecto.
- -On : Indicamos el nivel de optimización. Tenemos 0,1,2,3 y s.

Hay otras opciones interesantes dependientes de la arquitectura PIC que nos permiten alterar la forma en que el compilador convierte el código para el PIC. Con ellos tendremos más control sobre el código ensamblador generado.

- -mmore_call : Activa la pila software para almacenar la dirección en los saltos.
- -mno_share_memory : Indica al compilador que el microcontrolador no tiene memoria compartida.
- -mno_interrupt : Indica al compilador que no se genere código de tratamiento para interrupciones.
- -mp=<procesador>: Sirve para indicar el modelo de microcontrolador elegido.
- -mem-res=<tamaño>: Indica al compilador el tamaño de memoria RAM para reservar. Con este parámetro podemos (y debemos) reservar espacio para variables globales y estáticas definidas en ficheros distintos al principal.
- -mfreg-num=<n_reg>: Especifica el número de registros virtuales que se usarán. El máximo es 32 y el mínimo 4.
- -mstack-size=<tamaño>: Especifica el tamaño en bytes de la pila software. Por defecto es de 16 bytes.

1.2.2. GPUtils

Las principales utilidades de GPUtils son un ensamblador GPasm, un enlazador GPlink y un gestor de librerias GPlib.

Gpasm, el ensamblador de las gputils, nos permite pasar de un fichero en código ensamblador a uno en código objeto. Si el código no tiene símbolos sin instanciar el archivo objeto es usable directamente para programar el microcontrolador. En caso contrario necesitaremos pasar por el enlazador.

La sintaxis es:

```
gpasm [opciones] fila-asm
```

donde fila-asm identifica a un archivo de código ensamblador, normalmente con la extensión ".s" , pero también se suelen usar como extensiones ".asm".

Entre las opciones, las más importantes y útiles para el trabajo normal son:

- c : que genera código relocalizable.
- o <fila>: que establece un nombre alternativo al objeto compilado, que por defecto será el mismo que el dado como entrada, pero cambiando la extensión del mismo por ".o" si el código es relocalizable o por ".hex"si es absoluto.
- p <procesador>: que permite elegir para que microcontrolador se va a compilar el fichero. Esto también se puede hacer desde el código ensamblador. En nuestro caso será "-p 16F877".

GPLINK

Gplink es el enlazador para código relocalizable. Relocaliza y enlaza los archivos objeto que forman nuestra aplicación y crean un único archivo listo para transferir a la memoria de nuestro microcontrolador. Su sintaxis es:

```
gplink [opciones] [objetos] [librerías]
```

donde [objetos] es uno o más archivos objeto que forman nuestra aplicación. [librerías] es un argumento opcional para indicar las librerías que contienen rutinas necesarias para la compilación.

Entre las opciones, las más importantes y útiles para el trabajo normal son:

- I <directorio>: Especifica un directorio para incluir en la búsqueda de archivos. Puede haber varios usos en una misma llamada.

- o <fila>: establece un nombre alternativo al programa compilado, que por defecto será "a.hex".

- s <archivo>: especifica el archivo con las definiciones necesarias para el enlazador, correspondientes al microcontrolador usado. Este archivo (llamado 'linker script') indica la memoria disponible, su tipo y localización, y demás parámetros necesarios para que el enlazador haga su trabajo. Si no se especifica ningún archivo, gplink intentará usar el indicado en los códigos objeto y la ruta por defecto.

GPLIB

Gplib crea, modifica y extrae archivos objeto de librerías. Con esto podemos agrupar una colección de objetos relacionados en un único archivo, y pasar este único archivo a gplink. Gplink solo tomará los objetos que necesite de la librería, por lo que ahorramos en llamar a gplink con los mismos objetos, sin incrementar el tamaño del archivo final. Su sintaxis es:

```
gplib [opciones] librería [miembros]
```

donde librería es el nombre de la librería que se va a crear, modificar o simplemente inspeccionar. [miembros] son los archivos objeto que queremos extraer, borrar o insertar, según se indiquen en las opciones.

Entre las opciones, las más importantes y útiles para el trabajo normal son:

- c : crea una nueva librería.

- d : borra miembros de una librería.

- r : suma o reemplaza de una librería.

- s : muestra los símbolos globales de una librería.

- t : muestra los miembros de una librería.

- x : extrae objetos de una librería.

1.2.3. Programador

Necesitamos un sistema programador para trabajar con cualquier microcontrolador. Para los PIC de baja media existen numerosos sistemas en la web. Un sistema programador consta de dos partes, un programador software y uno hardware. Los dispositivos hardware suelen diferenciarse en la interfaz de conexión y los microcontroladores que soportan. Debemos elegir el programador adecuado a nuestras posibilidades, pero por menos de 3 euros podemos

construirnos uno. Una combinación funcional es un programador por puerto paralelo junto con odissey para Linux o ic-prog para Windows. Podemos encontrar más información en el apartado de documentación de la forja del proyecto.

1.2.4. Archivos auxiliares

Para que la programación sea posible necesitamos unos archivos extra. Por un lado la librería libgcc.a generada durante la creación del compilador C. Cuando enlacemos el código final debemos siempre especificar esta librería ya que contiene código necesario para la ejecución correcta de los programas en C. También deberemos tener el archivo de definiciones para el enlazador que corresponda a nuestro microcontrolador. Para el PIC16f877 este archivo es p16f877.lkr y contiene información necesaria para que el enlazador gplink pueda trabajar. En caso de no usar la instalación por defecto (e incluso así) debemos especificar su ruta o copiarlo al directorio adecuado para facilitar su operación.

Otro archivo no necesario pero sí muy recomendable es el archivo de cabecera dependiente del microcontrolador y que contiene las definiciones de las direcciones de los registros-fila del compilador así como los nombre particulares para los bits de registros-fila de configuración o estado. Para nuestro PIC es el archivo p16f877.h y deberemos incluirlo en el código C, preferentemente.

1.3. Atributos de función especiales

Junto con los distintos atributos usables para caracterizar funciones, en la arquitectura PIC, GCC ofrece un atributo extra. El atributo permite generar funciones interrupción, en las que se guarda todo el entorno anterior a la ejecución de la función para ser cargado después de la misma. Además también es posible especificar una función interrupción principal que establezca el vector interrupción apropiado en el mapa de memoria del PIC. La forma de especificar el atributo es:

tipo __attribute__ ((__interrupt__("main"))) funcion1 () ...

o

tipo __attribute__ ((__interrupt__)) funcion2 () ...

para funciones interrupción pero en las que no se desea fijar el vector.

La función interrupción con modificador main, será llamada cada vez que se produzca una interrupción en el microcontrolador. El programador debe averiguar la fuente de la interrupción generada y actuar en consecuencia. El esquema visto permite crear manejadores de interrupción a medida.

1.4. Atributos en el ensamblador en linea

La sintaxis de la función para introducir código ensamblador en las funciones C es:

```
__asm__ ("instrucciones" : lista_salida : lista_entrada : lista_destruida);
```

El patrón "instrucciones" será una cadena con las instrucciones en ensamblador para salida al fichero. Esta cadena contendrá tantos %n como se necesite, siendo n un número de 0 al que corresponda que representa el argumento de las listas que se desea establecer como salida. Las distintas listas estableceran los datos de salida, entrada o sobreescritos que se producirán con el código ensamblador, respectivamente. Las listas se crean indicando entre parentesis la variable o valor directo que ocuparan las posiciones del código, precedidos con una cadena que indique su tipo, "v" para registros, "i" para constantes enteras o "=v" para registros cuyo valor será sobreescrito. Las listas se separan con : a menos que de derecha a izquierda esten vacias.

Por ejemplo el código siguiente:

```
char dato1 = 1;
char dato2 = 2;
__asm__ ("movf %1,W\n\tmovwf %0\n":"=v" (dato1):"v" (dato2));
```

Copia el registro que contiene el valor dato2 en el registro que contiene dato1. Por lo tanto realiza la operación dato1 = dato2. No hemos indicado los últimos : para la lista de elementos destruidos porqué esta esta vacía, pero podríamos hacerlo perfectamente. El patrón %1 emparejará con el dato2 y esté estará en un registro gracias a indicar el atributo "v". El parámetro %0 emparejará con dato1, ya que en las listas está primero pese a usarse segundo en la cadena de salida ensamblador. Aquí se usa "=v" con el signo igual ya que el registro es de salida, y hemos de indicarle a GCC que su valor no se mantendrá tras la función.

Dado que GCC optimizará todo el código que no resulte de utilidad, si los valores de salida de una función en linea, no son usados GCC anulará la función __asm__. Para evitar esto debemos incluir el atributo __volatile__ en la llamada a la función para que GCC sepa que aunque los valores de salida no sean usados (o no haya valores de salida), el código introducido tiene efectos secundarios y no debe ser eliminado.

1.5. Más y actualizada información

Para conseguir la última información disponible podemos visitar:
www.pjmicrocontroladores.es