SISTEMA DE DESARROLLO GP_BOT

MANUAL DE PROGRAMACIÓN

Rev: 0.2

Guillermo González de Rivera Peces[1]
Carlos Jesús Venegas - Iván González Martínez[2]

21 de marzo de 2002

[1]Profesor de Robótica de la ETSI - UAM
[2]Miembros del Club de Robótica - Mecatrónica de la ETSI - UAM

Índice general

Índice de figuras

0.1. Prólogo

Este manual intenta ser una ayuda introductoria sobre la programación de la tarjeta GP_Bot desarrollada en la ETSI - UAM, con el fin de hacer más sencillo iniciarse en el desarrollo de aplicaciones de robótica autónoma con este sistema de desarrollo.

Todos los programas que aparecen en este manual se han desarrollado en ensamblador y han sido probados en el entorno IDE desarrollado por Motorola para el microcontrolador GP32 y que se puede encontrar en la dirección ....[1]Este manual no pretende, por tanto, ser un manual técnico ni de programación de los microcontroladores de Motorola de la familia 68HC08. Para obtener este tipo de información es aconsejable consultar la dirección .....

0.2. Notas de los autores

Parte del contenido de este manual (Arquitectura del MC68HC908GP32) se basa en una traducción del documento oficial de Motorola sobre el microcontrolador MC68HC908GP32. Para más detalles consultar el documento en la web de Motorola.

Para cualquier error o consulta pueden enviar un mail a Club.Mecatronica@ii.uam.es.

[1]Todas las direcciones incluidas en este prólogo hacen referencia a la Web de Motorola.

0.3. Revisión

REVISIÓN	DESCRIPCIÓN
0.1	Primera versión publicada.
0.2	Índice de Figuras. Control de velocidad de motores CC. Ejemplo SCI.

Capítulo 1

Introducción a la tarjeta GP_Bot

1.1. Introducción

Este sistema de desarrollo surge de la necesidad de contar con un sistema flexible, de propósito general y de cierta potencia para el desarrollo de las prácticas de la asignatura de Robótica Autónoma, impartida en la ETS de Informática de la Universidad Autónoma de Madrid.

Un buen sistema de desarrollo debe contar, por un lado, con los recursos suficientes para el control de los elementos básicos que puede necesitar lo que denominaríamos como un robot básico, tales como motores, sensores de infrarrojos, pulsadores, etc. Y por otro lado debe tener una gran potencia de cálculo así como gran cantidad de memoria para poder gestionar con éxito toda clase de complicados algoritmos de inteligencia artificial, creación de mapas de entorno, etc.

Lo primero es sencillo de conseguir pero lo segundo no tanto, se complica el diseño y se encarece bastante, con lo que sería difícil dotar a cada puesto de laboratorio de un sistema completo.

La solución que se ha adoptado finalmente cumple el primer objetivo claramente, como se verá a continuación, con la utilización de un moderno microcontrolador de Motorola, con bastantes herramientas de apoyo y completa documentación. Esto será suficiente para la mayoría de las aplicaciones.

El problema de la necesidad de mayor potencia se ha solucionado incluyendo un módulo de radio que permite una comunicación bidireccional con un estación base, que podrá ser un PC. De esta forma toda la potencia necesario la pondrá el PC y se limitará a dar órdenes sencillas de movimiento y peticiones de lecturas del entorno al robot.

1.2. Tarjeta GP_Bot

Capítulo 2

Arquitectura del MC68HC908GP32

2.1. Puertos de Entrada/Salida (I/O Ports)

El microcontrolador GP32 presenta 5 puertos que ofrecen un total de 33 pines bidireccionales, siendo posible configurar cada uno de ellos como entrada o como salida. Además, a los pines de los puertos A, C y D se les puede conectar una resistencia de pull-up que se pueden habilitar por software cuando se emplean como bits de entrada. En caso de configurarlos como salida, las resistencias de pull-up se deshabilitan automáticamente.

2.1.1. Puerto A

El puerto A está formado por ocho pines que están compartidos con el módulo de interrupciones de teclado (KBI). El puerto A dispone además de resistencias de pullup configurables por software si se configura como entrada.

2.1.1.1. Registro de datos

El registro de datos del puerto A (PTA) se encuentra mapeado en memoria en la dirección $0000.

PTA7	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0
KBD7	KBD6	KBD5	KBD4	KBD3	KBD2	KBD1	KBD0

2.1.1.2. Registro de dirección

El registro de dirección del puerto A (DDRA) determina la configuración de cada uno de los pines del puerto, ya sea como entrada o como salida. Para configurar como salida es necesario poner a 1 el bit correspondiente al pin que se quiere configurar como salida. Para configurar como entrada, se pone a 0 el bit correspondiente.

El registro de dirección del puerto A se encuentra mapeado en memoria en la dirección $0004.

DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0

2.1.1.3. Registro habilitador de pullup

El registro que habilita el pullup de entrada del puerto A (PTAPUE) contiene un dispositivo pullup configurable por software para cada uno de los pines del puerto A. Cada bit se puede configurar individualmente y requiere que el bit correspondiente del registro de dirección, DDRA, esté configurado como entrada. Cada pullup es automáticamente deshabilitado cuando el bit correspondiente del DDRA se configure como salida.

El registro que habilita el pullup de entrada del puerto A se encuentra mapeado en memoria en la dirección $000D.

PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0

2.1.2. Puerto B

El puerto B está formado por ocho pines que están compartidos con el conversor analógico/digital (ADC). No dispone de resistencias de pull-up.

2.1.2.1. Registro de datos

El registro de datos del puerto B (PTB) se encuentra mapeado en memoria en la dirección $0001.

PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0
AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

2.1.2.2. Registro de dirección

El registro de dirección del puerto B (DDRB) determina la configuración de cada uno de los pines del puerto, ya sea como entrada o como salida. Para configurar como salida es necesario poner a 1 el bit correspondiente al pin que se quiere configurar como salida. Para configurar como entrada, se pone a 0 el bit correspondiente.

El registro de dirección del puerto B se encuentra mapeado en memoria en la dirección $0005.

DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0

2.1.3. Puerto C

El puerto C está formado por siete pines de propósito general. El puerto C dispone además de resistencias de pullup configurables por software si se configura como entrada.

2.1.3.1. Registro de datos

El registro de datos del puerto C (PTC) se encuentra mapeado en memoria en la dirección $0002.

	PTC6	PTC5	PTC4	PTC3	PTC2	PTC1	PTC0

2.1.3.2. Registro de dirección

El registro de dirección del puerto C (DDRC) determina la configuración de cada uno de los pines del puerto, ya sea como entrada o como salida. Para configurar como salida es necesario poner a 1 el bit correspondiente al pin que se quiere configurar como salida. Para configurar como entrada, se pone a 0 el bit correspondiente.

El registro de dirección del puerto C se encuentra mapeado en memoria en la dirección $0006.

	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0

2.1.3.3. Registro habilitador de pullup

El registro que habilita el pullup de entrada del puerto C (PTCPUE) contiene un dispositivo pullup configurable por software para cada uno de los pines del puerto C. Cada bit se puede configurar individualmente y requiere que el bit correspondiente del registro de dirección, DDRC,

esté configurado como entrada. Cada pullup es automáticamente deshabilitado cuando el bit correspondiente del DDRC se configure como salida.

El registro que habilita el pullup de entrada del puerto C se encuentra mapeado en memoria en la dirección $000E.

	PTCPUE6	PTCPUE5	PTCPUE4	PTCPUE3	PTCPUE2	PTCPUE1	PTCPUE0

2.1.4. Puerto D

El puerto D está formado por ocho pines de los cuales cuatro están compartidos con el módulo de interfaz serie con periféricos (SPI) y los otros cuatro con los dos módulos de interfaz del temporizador (TIM). El puerto D dispone además de resistencias de pullup configurables por software si se configura como entrada.

2.1.4.1. Registro de datos

El registro de datos del puerto D (PTD) se encuentra mapeado en memoria en la dirección $0003.

PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
T2CH1	T2CH0	T1CH1	T1CH0	SPSCK	MOSI	MISO	SS

2.1.4.2. Registro de dirección

El registro de dirección del puerto D (DDRD) determina la configuración de cada uno de los pines del puerto, ya sea como entrada o como salida. Para configurar como salida es necesario poner a 1 el bit correspondiente al pin que se quiere configurar como salida. Para configurar como entrada, se pone a 0 el bit correspondiente.

El registro de dirección del puerto D se encuentra mapeado en memoria en la dirección $0007.

DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0

2.1.4.3. Registro habilitador de pullup

El registro que habilita el pullup de entrada del puerto D (PTDPUE) contiene un dispositivo pullup configurable por software para cada uno de los pines del puerto D. Cada bit se puede configurar individualmente y requiere que el bit correspondiente del registro de dirección, DDRD, esté configurado como entrada. Cada pullup es automáticamente deshabilitado cuando el bit correspondiente del DDRD se configure como salida.

El registro que habilita el pullup de entrada del puerto D se encuentra mapeado en memoria en la dirección $000F.

PTDPUE7	PTDPUE6	PTDPUE5	PTDPUE4	PTDPUE3	PTDPUE2	PTDPUE1	PTDPUE0

2.1.5. Puerto E

El puerto E está formado por dos pines que están compartidos con el módulo de interfaz de comunicación serie (SCI). No dispone de resistencias de pull-up.

2.1.5.1. Registro de datos

El registro de datos del puerto E (PTE) se encuentra mapeado en memoria en la dirección $0008.

						PTE1	PTE0
						RxD	TxD

2.1.5.2. Registro de dirección

El registro de dirección del puerto E (DDRE) determina la configuración de cada uno de los pines del puerto, ya sea como entrada o como salida. Para configurar como salida es necesario poner a 1 el bit correspondiente al pin que se quiere configurar como salida. Para configurar como entrada, se pone a 0 el bit correspondiente.

El registro de dirección del puerto E se encuentra mapeado en memoria en la dirección $000C.

						DDRE1	DDRE0

2.1.6. Ejemplos de utilización

Se pueden encontrar ejemplos de utilización en:

- Sección 5.3

2.2. Módulo de Interfaz del Temporizador (TIM)

El TIM es un temporizador de dos canales que proporciona una referencia temporal a los capturadores, comparadorres y a las funciones de PWM (Pulse-Width-Modulation). En el caso particular de este microcontrolador, existen dos modulos a los que llamaremos TIM1 y TIM2.

Cada uno de los TIM incluye:

- Dos canales que pueden actuar como capturadores o comparadores:
  - Disparo del capturador en flanco de subida, en flanco de bajada o en ambos flancos.
  - Comparador con salida en alto, bajo o alternada.
- Generación de señal PWM con buffer o sin buffer.
- Entrada de reloj del TIM programable con selección de siete posibles frecuencias.
- El contador puede operar como contador libre o con módulo.
- Intercambio del valor del pin si overflow.
- Contador del TIM con bits de stop y reset.
- Arquitectura modular expandible a ocho canales.

Los pines de entrada / salida del TIM se nombran como:

	T[1,2]CH0	T[1,2]CH1
TIM1	PTD4 / T1CH0	PTD5 / T1CH1
TIM2	PTD6 / T2CH0	PTD7 / T2CH1

2.2.1. Descripción funcional

El componente central del TIM es un contador de 16 bits que opera como contador libre o con módulo. Este contador provee la referencia temporal al resto de componentes. Los registros de módulo del contador TMODH:TMODL, controlan el valor del módulo del contador. Por software es posible leer el valor del contador en cualquier momento sin que ello afecte a la cuenta.

Los dos canales (por temporizador) se pueden programar independientemente como capturadores o comparadores. Si un canal se configura como capturador, entonces el pullup interno debería habilitarse para ese canal.

2.2.2. Preescalado del contador

El reloj que emplea el TIM puede seleccionarse de las siete distintas salidas preescaladas. El preescalador genera siete rangos de reloj a partir del reloj de bus interno. Los bits de selección de preescalado PS[2:0], en el registro de estado y control del TIM seleccionan la fuente de reloj.

2.2.3. Capturador

La función de capturador permite al TIM capturar el tiempo en que ocurre un evento externo. Cuando ocurre un flanco activo en el pin del canal de captura, el TIM almacena el contenido del contador del TIM en los registros del canal, TCHxH:TCHxL. El tipo de flanco de captura es programable. El capturador puede generar una petición de atención a interrupción.

2.2.4. Comparador

La función de comparador permite al TIM generar pulsos periodicos con polaridad, duración y frecuencia programables. Cuando el contador alcanza el valor de los registros de un canal comparador, el TIM puede poner a uno, a cero o bien intercambiar el valor del pin del canal. El comparador puede generar una petición de atención a interrupción.

2.2.4.1. Comparación de salida no buffereada

2.2.4.2. Comparación de salida buffereada

2.2.5. Modulación por ancho de pulso (PWM)

Si se usan las características de alternar el valor de un pin cuando existe overflow con un canal configurado como comparador, el TIM puede general señales PWM. El valor de los registros de módulo del contador determinan el periodo de la señal PWM. El pin del canal alterna cuando el contador alcanza el valor de los registros de módulo. El tiempo entre overflows es el periodo de la señal PWM.

El valor del comparador en los registros del canal del TIM determina el ancho de pulso de la señal PWM. El tiempo entre overflow y el comparador es el ancho del pulso. Programar el TIM para que ponga a cero el pin del canal del comparador si el estado del pulso PWM es un uno. Programar el TIM para poner a uno el pin si el estado de la señal de PWM es un cero.

El valor de los registros del modulo del contador y la salida preescalada determinan la salida PWM. La frecuencia de una señal PWM de 8 bits es variable en 256 incrementos. Escribiendo $00FF (255) en los registros del modulo del contador se produce un periodo PWM de 256 veces el periodo del reloj de bus interno si se selecciona un preescalado de valor $000.

El valor de los registros del canal determinan el ancho de la salida PWM. El ancho del pulso de una señal PWM de 8 bits varía en 256 incrementos. Escribiendo $0080 (128) en los registros del canal se produce un ciclo de trabajo de 128/256 o 50 %.

2.2.5.1. Generación de señal PWM no buffereada

2.2.5.2. Generación de señal PWM buffereada

2.2.5.3. Inicialización del PWM

Para asegurar una correcta operación cuando se generan señales PWM buffereadas o no buffereadas, se usa el siguiente procedimiento de inicialización:

1. En el registro de estado y control del TIM (TSC):
   - *a*) Parar el contador del TIM poniendo un uno en el bit de stop, TSTOP.
   - *b*) Resetear el contador del TIM poniendo a uno el bit de reset, TRST.

2. En los registros del modulo del contador (TMODH:TMODL), escribir el valor del periodo de la señal PWM requerida.
3. En los registros del canal x (TCHxH:TCHxL), escribir el valor del ancho de pulso requerido.
4. En el registros de estado y control del canal x (TSCx):
   a) Escribir 0:1 (para comparación o señal PWM no buffereada) or 1:0 (para comparación o señal PWM buffereada) en los bits de selección de modo, MSxB:MSxA.
   b) Escribir 1 en el bit de intercambiar si overflow (toggle-to-overflow), TOVx.
   c) Escribir 1:0 (para poner a cero la salida del comparador) o 1:1 (para poner a uno la salida del comparador) en los bits de selección de flanco, ELSxB:ELSxA. La acción de salida si el comparador se activa debe forzar la salida al nivel contrario del ancho de pulso.
5. En el registro de estado y control del TIM (TSC), poner a cero el bit de stop, TSTOP.

IMPORTANTE: En la generación de señal PWM, no se programa el canal PWM para intercambiar si se produce la comparación.

2.2.6. Interrupciones

Las siguientes fuentes del TIM pueden provocar interrupciones:

- Flag de overflow del TIM (TOF) - El bit TOF se pone a uno cuando el valor del contador vuelve a $0000 después de igualar el valor de los registros del módulo del contador. El bit de interrupción de overflow, TOIE, habilita la petición de atención a la interrupción CPU del overflow del TIM. TOF y TOIE están en el registros de estado y control del TIM.
- Flag de canal del TIM (CH1F:CH0F) - El bit CHxF se pone a uno cuando se produce una captura o una comparación en el canal x. La petición de interrupción CPU es controlada por el bit de habilitación de interrupción del canal x, CHxIE. La petición de interrupción de CPU del TIM esta habilitada cuando CHxIE = 1. CHxF y CHxIE están en el registro de estado y control del canal x del TIM.

2.2.7. Señales I/O

El puerto D comparte cuatro de sus pines con el TIM. Los cuatro pines de los canales I/O del TIM son T1CH0, T1CH1, T2CH0 y T2CH1. Cada pin de canal I/O es programable independientemente como capturador o como comparador. T1CH0 y T2CH0 pueden configurar el canal como comparador buffereado o PWM buffereado.

2.2.8. Registros I/O

IMPORTANTE: La referencia es tanto para el temporizador 1 como para el temporizador 2, omitiendo el número del temporizador.

Estos registros I/O controlan y monitorizan la operación del TIM:

- Registro de estado y control (TSC).
- Registros del contador (TCNTH:TCNTL).
- Registros del módulo del contador (TMODH:TMODL).
- Registros de estado y control del canal (TSC0, TSC1).
- Registros del canal (TCH0H:TCH0L, TCH1H:TCH1L).

2.2.8.1. Registro de estado y control del TIM

El registro de estado y control del TIM (TSC):

- Habilita las interrupció de overflow.
- Flag de overflow.
- Para el contador.
- Resetea el contador.
- Preescala el reloj del contador.

La dirección de este registro es $0020 (T1SC) y $002B (T2SC).

TOF	TOIE	TSTOP	TRST		PS2	PS1	PS0

TOF Bit de Flag de Overflow

Este flag de lectura/escritura se pone a uno cuando el contador resetea a $0000 después de alcanzar el valor del módulo programado en los registros del modulo del contador. Se limpia el TOF leyendo el registro de estado y control cuando el TOF esté a uno y entonces escribir un cero en el TOF. Si otro overflow del TIM ocurre antes de completar la secuencia de limpiado, escribir un cero en el TOF no tendrá efecto. De este modo, una petición de interrupción del TOF no se puede perder debido a una limpieza inadvertida.

Reset limpia el bit de TOF. Escribir un uno lógico en el TOF no tiene efecto.
1 = El contador del TIM ha alcanzado su valor de modulo.
0 = El contador del TIM no ha alcanzado su valor de módulo.

TOIE Bit de Habilitación de Interrupción de Overflow

Este bit de lectura/escritura habilita la interrupción de overflow del TIM cuando el bit TOF se pone a uno.

Reset pone a cero este bit.
1 = Habilita la interrupción de overflow del TIM.
0 = Deshabillita la interrupción de overflow del TIM.

TSTOP Bit de Stop

Este bit de lectura/escritura para el contador del TIM. El contador se inicia de nuevo cuando TSTOP se pone a cero.

Reset pone a uno el bit TSTOP, parando el contador del TIM hasta que el software lo ponga a cero.
1 = Parar el contador del TIM.
0 = Contador del TIM activo.

TRST Bit de Reset

Poniendo este bit de solo escritura a uno se resetea el contador y el preescalado del TIM. Poner a uno este registro no afecta a ningún otro registro. El contado comienza en $0000. TRST se pone automáticamente a cero después de resetear el contador y siempre se lee un cero lógico.

Reset pone a cero este bit.
1 = Preescalado y contador del TIM a cero.
0 = No tiene efecto.

PS[2:0] Bits de Selección del Preescalado

Estos bits de lectura/escritura seleccionan una de las siete salidas preescaladas como entrada del contador del TIM.

Reset pone a cero todos los bits.

PS2	PS1	PS0	Fuente de reloj del TIM
0	0	0	Reloj de bus interno / 1
0	0	1	Reloj de bus interno / 2
0	1	0	Reloj de bus interno / 4
0	1	1	Reloj de bus interno / 8
1	0	0	Reloj de bus interno / 16
1	0	1	Reloj de bus interno / 32
1	1	0	Reloj de bus interno / 64
1	1	1	No disponible

2.2.8.2. Registros del contador del TIM

Los dos registros de solo lectura del contador del TIM contiene el byte más significativo y el menos significativo del valor del contador. Leer el byte más significativo (TCNTH) captura el byte menos significativo (TCNTL) en un buffer. Lecturas consecutivas del TCNTH no afectan al valor capturado del TCNTL hasta que éste sea leído.

Reset pone a cero los registros del contador del TIM. Poniendo a uno el bit de reset (TRST) también se ponen a cero los registros del contador.

Las direcciones de los registros del contador para el temporizador 1 son $0021 (T1CNTH) y $0022 (T1CNTL).

Las direcciones de los registros del contador para el temporizador 2 son $002C (T2CNTH) y $002D (T2CNTL).

2.2.8.3. Registros de módulo del contador del TIM

Los registros de lectura/escritura del módulo del contador del TIM contienen el valor del módulo para el contador del TIM. Cuando el contador alcanza el valor del módulo, el flag de overflow (TOF) se pone a uno, y el contador del TIM comienza su cuenta en $0000 al siguiente ciclo de reloj. Escribir el byte más significativo (TMODH) inhibe el bit TOF y la interrupción de overflow hasta que se escriba el byte menos significativo (TMODL).

Reset pone a uno todos los bits de los registros del módulo del contador.

Las direcciones de los registros del módulo del contador para el temporizador 1 son $0023 (T1MODH) y $0024 (T1MODL).

Las direcciones de los registros del módulo del contador para el temporizador 2 son $002E (T2MODH) y $002F (T2MODL).

2.2.8.4. Registros de estado y control de canal del TIM

En cada uno de los registros de estado y control de canal del TIM:

- Flag de captura o comparación.
- Habilitar la interrupción de captura o comparación.
- Selección de capturador, comparación u operación PWM.
- Selección de salida en alto, en bajo o altenar salida en la comparación.
- Selección de flanco de subida, de bajada o cualquier flanco para activar la captura de entrada.
- Selección de alternar la salida en caso de overflow.
- Selección de un ciclo PWM de 100 % de ciclo de trabajo.

- Selección de comparación u operación PWM buffereada o no buffereada.

El registro de estado y control para el canal 0 del temporizador 1 se encuentra en la dirección $0025 (T1SC0).

El registro de estado y control para el canal 0 del temporizador 2 se encuentra en la dirección $0030 (T2SC0).

CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	TOV0	CH0MAX

El registro de estado y control para el canal 1 del temporizador 1 se encuentra en la dirección $0028 (T1SC1).

El registro de estado y control para el canal 1 del temporizador 2 se encuentra en la dirección $0033 (T2SC1).

CH1F	CH1IE		MS1A	ELS1B	ELS1A	TOV1	CH1MAX

CHxF Bit de Flag del Canal x

Cuando el canal x está configurado como capturador, este bit de lectura/escritura se pone a uno cuando se produce un flanco activo en el pin del canal x. Cuando el canal x está configurado como comparador, CHxF se pone a uno cuando el valor de los registros del contador del TIM coincide con el valor de los registros del canal x.

Cuando la solicitud de interrupción está habilitada (CHxEI = 1), se pone a cero el CHxF leyendo el registro de estado y control del canal x cuando CHxF está a uno y escribiendo entonces un cero en el CHxF. Si otra solicitud de interrupción ocurre antes de que se complete la secuencia de puesta a cero, entonces escribir un cero en el bit CHxF no tendrá efecto. De este modo, ninguna solicitud de interrupción se pierde por una inoportuna puesta a cero del CHxF.

Reset pone a cero el bit CHxF. Escribir un uno en el CHxF no tiene efecto.

1 = Captura o comparación en el canal x.

0 = No hay captura o comparación en el canal x.

CHxIE Bit de Habilitación de Interrupción del Canal x

Este bit de lectura/escritura habilita el servicio de solicitud de interrupción CPU del TIM en el canal x.

Reset pone a cero el bit CHxIE.

1 = Canal x con solicitud de interrupción de CPU habilitada.

0 = Canal x con solicitud de interrupción de CPU inhabilitada.

MSxB Bit B de Selección de Modo

Este bit de lectura/escritura selecciona la comparación/operación PWM buffereada. MSxB solo existe en el registro de estado y control del canal 0 de ambos temporizadores TIM1 y TIM2.

Poner a uno el MSxB dehabilita el registro de estado y control del canal 1 y pasa TCH1 a E/S de propósito general.

Reset pone a cero el bit MSxB.

1 = Habilitación del comparador/operacion PWM buffereada.

0 = Deshabilita el comparador/operación PWM bufferada.

MSxA Bit A de Selección de Modo

Cuando ELSxB:A != 00, este bit de lectura/escritura selecciona entre capturador o comparador/operación PWM sin bufferear.

1 = Comparador de salida/operación PWM sin bufferear.

0 = Operación de captura.

Cuando ELSB:A = 00, este bit de lectura/escritura selecciona el nivel inicial de salida del pin TCHx.

Reset pone a cero el bit MSxA.

1 = Nivel de salida inicial bajo.

0 = Nivel de salida inicial alto.

IMPORTANTE: Antes de configurar los bits MSxA y MSxB es importante poner a uno los bits TSTOP y TRST en el registro de estado y control del TIM (TSC).

ELSxB:ELSxA Bits de Selección de Flanco/Nivel

Cuando el canal x es un capturador, estos bits de lectura/escritura controlan el flanco activo del canal x.

Cuando el canal x es un comparador, ELSxB y ELSxA controlan el comportamiento de salida del canal x cuando ocurre una comparación.

Cuando ELSxB y ELSxA son ambos cero, el canal x está conectado al puerto D y el pin PTDx/TCHx está disponible como pin de E/S de propósito general.

Reset poner a cero ambos bits.

En la siguiente tabla se muestra como funcionan los bits ELSxB y ELSxA:

MSxB:MSxA	ELSxB:ELSxA	Modo	Configuración
X0	00	E/S	Bit Puerto D - Nivel alto
X1	00		Bit Puerto D - Nivel bajo
00	01	Capturador de entrada	Solo flanco de subida
00	10		Solo flanco de bajada
00	11		Flanco de subida y bajada
01	01	Comparador de salida o PWM	Alternar salida
01	10		Salida a cero
01	11		Salida a uno
1X	01	Comparador de salida/PWM buffereado	Alternar salida
1X	10		Salida a cero
1X	11		Salida a uno

TOVx Bit de Intercambio si Overflow

Cuando el canal x es un comparador, este bit de lectura/escritura controla el comportamiento de la salida del canal x cuando el contador del TIM sufre overflow. Cuando el canal x es un capturador, no tiene efecto.

Reset pone a cero el TOVx.

1 = El pin del canal x cambia cuando el contador del TIM sufre overflow.

0 = El pin del canal x no cambia cuando el contador del TIM sufre overflow.

IMPORTANTE: Cuando el pin TOVx está a uno, el overflow del contador del TIM precede al canal comparador si ambos ocurren al mismo tiempo.

CHxMAX Bit de Ciclo de Trabajo Máximo del Canal

Cuando el bit TOVx está a cero, si el bit CHxMAX se pone a uno se fuerza el ciclo de trabajo de la señal PWM buffereada o no buffereada al 100 %.

2.2.8.5. Registros del canal del TIM

Estos registros de lectura/escritura contienen el valor del contador del TIM capturado por la función del capturador o el valor del comparador de la función del comparador.

El estado de los registros de canal del TIM después de un reset se desconocen.

Las direcciones para el TIM1 son:

- Para el canal 0 en $0026 (T1CH0H) y $0027 (T1CH0L).
- Para el canal 1 en $0029 (T1CH1H) y $002A (T1CH1L).

Las direcciones para el TIM2 son:

- Para el canal 0 en $0031 (T2CH0H) y $0032 (T2CH0L).
- Para el canal 1 en $0034 (T2CH1H) y $0035 (T2CH1L).

2.3. Conversor Analógico/Digital (ADC)

El microcontrolador MC68HC908GP32 dispone de un conversor A/D de 8 bits que puede ser usado por cualquiera de los 8 canales disponibles. Presenta modos de operación continuo y único, señalizados por un flag de finalización. El reloj empleado por el conversor es seleccionable (reloj de bus interno o reloj externo).

2.3.1. Pines del puerto ADC

Los pines PTB7/AD7 - PTB0/AD0 son pines de E/S de propósito general que están compartidos con los canales del ADC. Los bits de selección de canal definen que pines del puerto ADC se usan como señales de entrada. El ADC fuerza los pines de los canales seleccionados a que actúen como entrada. El resto de pines se pueden emplear como E/S de propóstico general. Escribir en el puerto o DDR no tendrá ningún efecto en los pines seleccionados para el ADC. La lectura de estos pines devuelve siempre 0.

2.3.2. Conversión de voltaje

Los voltajes de conversión van entre +5V ($FF) y 0V ($00).

2.3.3. Tiempo de conversión

La conversión comienza después de escribir en el ADSCR. Una conversión puede tomar entre 16 y 17 ciclos de reloj del ADC. Los bits ADIVx y ADICLK deben proveer una frecuencia de reloj del ADC de 1MHz. Luego el tiempo de conversión es igual a 16 o 17 ciclos de reloj del ADC entre la frecuencia del ADC (el número de ciclos de bus es igual al tiempo de conversión por la frecuencia de bus).

2.3.4. Conversión

El el modo de conversión continuo, el registro de datos del ADC se rellena con un nuevo dato después de cada conversión, sobreescribiendo el dato de la conversión anterior independientemente de que haya sido leído o no. Las conversiones continuan hasta que el bit ADCO sea puesto a cero. El bit COCO se pone a uno después de la primera conversión y permanece así hasta la siguiente escritura del registro de estado y control o la siguiente lectura del registro de datos del ADC.

En el modo de conversión única, la conversión comienza con la escritura del ADSCR. Solo se produce una conversión entre escrituras del ADSCR

2.3.5. Interrupciones

Cuando el bit AIEN se pone a uno, el módulo ADC es capaz de generar una interrupción de CPU después de cada conversión. La interrupcuón se genera si el bit COCO está a cero. El bit COCO no se usa como indicador de conversión completa cuando las interrupciones están habilitadas.

2.3.6. Registros de E/S

Los siguientes registros controlan y monitorizan la operación del ADC:

- Registro de estado y control del ADC (ADSCR).
- Registro de dato del ADC (ADR).
- Registro de reloj del ADC (ADCLK).

2.3.6.1. Registro de estado y control del ADC

COCO	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0

COCO Conversión Completa

Cuando el bit AIEN es cero, este bit de solo lectura se pone a uno cada vez que se completa una conversión, excepto el modo de conversión continuo, donde se pone a uno después de la primera conversión. Este bit se pone cero cuando se escribe el ADSCR o cuando se lee el ADR.
1 = Conversión completa (AIEN = 0).
0 = Conversión no completada (AIEN = 0)/Interrupción de CPU(AIEN = 1).

AIEN Bit de habilitación de interrupción del ADC

Cuando este bit es puesto a uno se genera una interrupción al final de una conversión del ADC. La señal de interrupción se pone a cero cuando el registro de dato es leído o el registro de estado y control es escrito.
Reset pone a cero el bit AIEN.
1 = Interrupción del ADC habilitada.
0 = Interrupción del ADC deshabilitada.

ADCO Bit de conversión continua del ADC

Cuando está a uno, el ADC convierte la muestras continuamente y actualiza el registro ADR al final de cada conversión. Cuando está a cero solo se completa una conversión entre cada escritura del ADSCR.
Reser pone a cero el bit ADCO.
1 = Conversión continua del ADC.
0 = Una conversión del ADC.

ADCH4-ADCH0 Bits de selección de canal del ADC

ADCH4-ADCH0 forman un campo de cinco bits que se emplea para seleccionar uno de los 16 canales del ADC. Solo ocho de esos canales están disponibles en esta MCU. El subsistema del ADC s apaga cuando todos los bits están a uno. Esto permite reducir el consumo cuando el ADC no se está empleando.

ADCH4	ADCH3	ADCH2	ADCH1	ADCH0	Entrada seleccionada
0	0	0	0	0	PTB0/AD0
0	0	0	0	1	PTB1/AD1
0	0	0	1	0	PTB2/AD2
0	0	0	1	1	PTB3/AD3
0	0	1	0	0	PTB4/AD4
0	0	1	0	1	PTB5/AD5
0	0	1	1	0	PTB6/AD6
0	0	1	1	1	PTB7/AD7
0 | 1	1 | 1	0 | 1	0 | 0	0 | 0	Reservados
1	1	1	0	1	V_{REFH}
1	1	1	1	0	V_{REFL}
1	1	1	1	1	ADC Apagado

2.3.6.2. Registro de dato del ADC

Se trata de un registro de resultado de ocho bits. Este registro se actualiza cada vez que se completa una conversión.

AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

2.3.6.3. Registro de reloj del ADC

El registro de reloj del ADC selecciona la frecuencia de reloj para el ADC.

ADIV2	ADIV1	ADIV0	ADICLK				

ADIV2-ADIV0 Bits de preescalado del reloj del ADC

ADIV2-ADIV0 forman un campo de tres bits que se emplea para seleccionar el ratio de división usado por el ADC para general su reloj interno. El reloj del ADC debería estar establecido en aproximadamente 1MHz.

ADIV2	ADIV1	ADIV0	Ratio del reloj del ADC
0	0	0	Entrada de reloj del ADC / 1
0	0	1	Entrada de reloj del ADC / 2
0	1	0	Entrada de reloj del ADC / 4
0	1	1	Entrada de reloj del ADC / 8
1	X	X	Entrada de reloj del ADC / 16

ADICLK Bit de selección de entrada de reloj del ADC

ADICLK selecciona entre el reloj de bus o el CGMXCLK como fuente de entrada de reloj para generar el reloj interno del ADC.

Reset selecciona el CGMXCLK como fuente de reloj del ADC.

Si el reloj externo (CGMXCLK) es igual o mayor que 1MHz, CGMXCLK puede ser usado como fuente de reloj del ADC. Si CGMXCLK es menor de 1MHz, usar reloj de bus generado por el PLL como fuente de reloj. Tan pronto como el reloj interno del ADC es aproximadamente 1MHz se puede garantizar una correcta operación.

1 = Reloj de bus interno.

0 = Reloj externo (CGMXCLK).

Frecuencia de reloj de entrada del ADC/ADIV2-ADIV0 = 1MHz

2.3.7. Ejemplos de utilización

Se pueden encontrar ejemplos de utilización en:

- Sección 5.4

2.4. Interfaz de Comunicación Serie (SCI)

2.5. Módulo de Interfaz Serie con Periféricos (SPI)

2.6. Módulo de Interrupciones de Teclado (KBI)

Capítulo 3

Entorno de desarrollo para el microcontrolador M68HC908GP32

3.1. Introducción

En la web de Motorola se dispone de un completo entorno de desarrollo para la familia de microcontroladores 68HC08. En el caso particular del micro MC68HC908GP32, procedemos a dar un sencillo tutorial de su manejo.

El entorno de desarrollo está compuesto por un conjunto de herramientas independientes, cada una de las cuales tiene una función específica dentro de los distintos pasos que hay que cubrir para llevar a cabo la programación correcta del microcontrolador. Estas partes son: Editor, Ensamblador/Compilador, Simulador In-Circuit, Programador, Debugger In-Circuit y 2º Simulador In-Circuit.

Como herramientas externa adicional se necesita la placa GP_Mon, incluida en el sistema de desarrollo GP_Bot, a la que conoceremos a partir de ahora como cable monitor.

3.2. Descripción general

Este entorno de desarrollo permite editar código, ensamblarlo, simularlo, programarlo en la memoria flash del micro y realizar un debug en tiempo real.

- El entorno de edición es donde tiene lugar el proceso de edición y ensamblado. Simplemente pulsando la tecla F4 o el botón de la barra de tareas, se ensamblará el fichero actualmente seleccionado. Los errores aparecerán destacados en el fichero.
- La simulación In-Circuit es una herramienta que permite simular el 68HC908GP32. Todas las instrucciones, interrupciones, y periféricos se pueden simular. Cuando se ejecuta, la simulación de I/O ocurre en el propio entorno. Cualquier entrada al micro es leída por el entorno,

y cualquier salida simulada es conducida al micro. Por ejemplo, si el código emplea el conversor A/D, y lee cualquiera de los canales, el voltaje mostrado en el simulador se corresponde con el que se encuentra en el pin correspondiente. Hay que tener en cuenta que el código nunca se ejecuta en el micro, solamente las entradas y salidas se toman del mundo real a traves del micro. Igualmente es importante conocer que la simulación se lleva a cabo más lentamente que una ejecución real. La principal ventaja del simulador es que pueden chequear los algoritmos, y monitorizar, ciclo a ciclo, las operaciones del código y los periféricos. La tecla F6 ejecuta el simulador In-Circuit.

- Si se quiere ejecutar el código en tiempo real, se dispone de dos posibilidades: cargarlo en la RAM interna o programar la flash.
- El programador de flash permite programar la flash con un fichero objeto. Además, se puede borrar, chequear si está en blanco y verificar y actualizar el contenido. El programador de flash es un programador de propósito general que permite programar cualquier micro 68HC08 con memoria flash on-chip. Cuando el programador se inicia, pregunta por el algoritmo que se desea usar durante la programación. Para el GP32 el apropiado es 908_GP32.08P. Posteriormente, solo resta seleccionar el fichero objeto que se quiere programar, usando el comando SS, y elegir "Program Module". La tecla F7 ejecuta el programador de flash.
- El debugger In-Circuit tiene una intefaz muy similar en apariencia y comandos al simulador In-Circuit. La principal diferencia está en que se está ejecutando en el micro y no se trata de un modelo simulado. Existen varias restricciones que hay que conocer antes de realizar un debug en tiempo real, y por tanto es necesario, antes de proceder, leer con cuidado la sección de limitaciones del fichero de ayuda. Teniendo en cuenta estás limitaciones, se pueden hacer muchas cosas en el debugger In-Circuit: cargar el código en la RAM, poner todos los breakpoints software que se desee y un único breakpoint hardware (en ejecución flash). La principal ventaja es que el código se ejecuta completamente a 2.4576MHZ (velocidad del bus). La tecla F8 ejecuta el debugger In-Circuit.

El contenido de esta sección se corresponde con una traducción propia del fichero WELCOME_GP.ASM que puede encontrarse en el directorio donde se instaló el programa.

3.3. El cable monitor (MON08)

La programación del microcontrolador se realiza a través del llamado "cable monitor", también conocido como MON08.

El MON08 contiene distintas partes:

- Una conexión al puerto serie del PC.
- Una placa con la circuitería necesaria para establecer la comunicación con el microcontrolador.
- Una conexión plana con los pines correspondientes del microcontrolador. Esta conexión depende de la implementación del conector de la placa.

Al igual que otros conocidos microcontroladores de Motorola, la familia 68HC08 dispone de un programa monitor en ROM para programar el micro. La funcionalidad de este programa es realizar operaciones sobre el mapa de memoria como puede ser escritura, lectura, etc. Con este programa se comunica el cable monitor.

En función del número de cables conectados al puerto serie del PC y al tipo de MON08 se puede resetear el micro por software, e incluso, cortarle la alimentación. Ambas operaciones son necesarias cuando se quiere programar la flash del micro.

Otro apartado interesante en la programación de la flash del micro es la existencia de unos bytes de seguridad que se deben suministrar al programa cuando este intenta acceder a la flash. Estos bits impiden el acceso a la información de la flash.

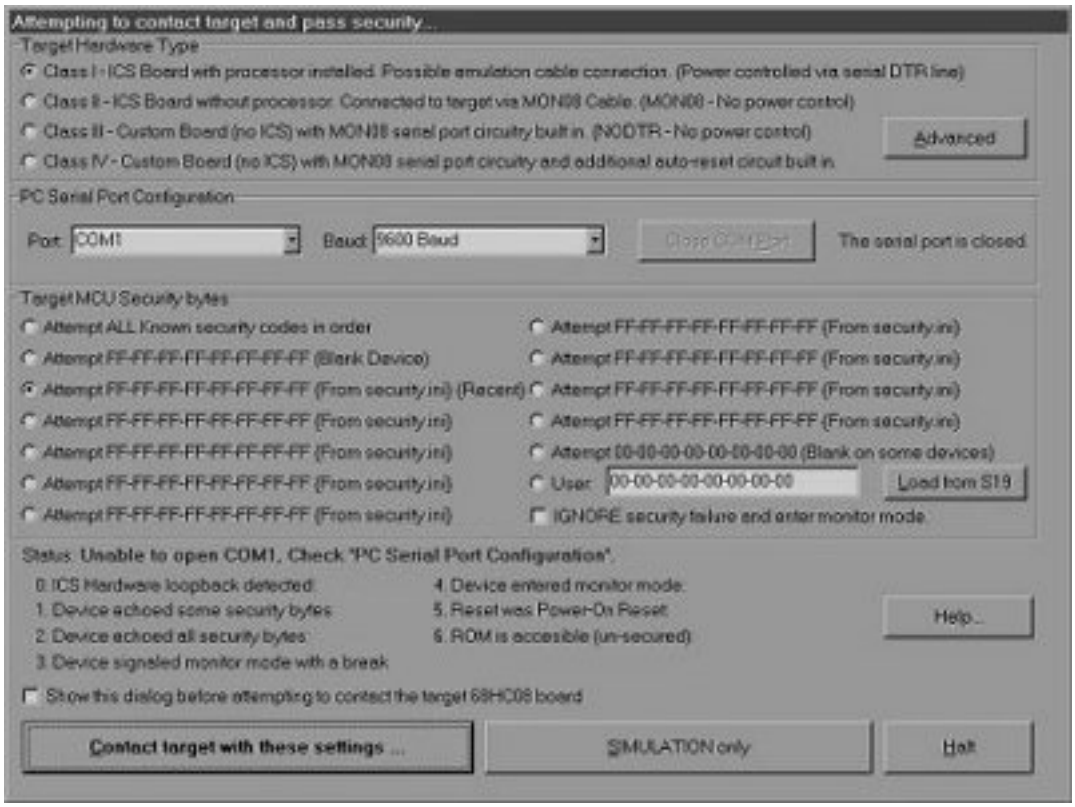

Figura 3.1: Ventana de acceso al microcontrolador

3.4. Acceso al microcontrolador a través del MON08

Algunas de las aplicaciones del entorno necesitan conectarse al microntrolador. Para ello emplean el MON08 para comunicarse con el microcontrolador. Este es el caso de la simulación, programación y debugger.

En todos los casos, lo primero que hace el entorno es establecer contacto con el microcontrolador y enviarle unos bytes de seguridad, los cuales corresponde con valores que deberían estar almacenados en ciertas posiciones de la memoria flash, normalmente, se corresponden con una parte de los vectores de interrupción.

Evidentemente, si es la primera vez que se programa el micro, éste se encuentra vacio, por defecto debe tener los valores FF o bien 00 en esas posiciones. El entorno probará con los valores que le suministremos, o bien, probará con los valores antes empleados, y que almacena en un fichero .log.

Si por cualquiera circunstancia, no disponemos de los bytes de seguridad, no seremos capaces de acceder al micro. La única solución, es acceder a la opción de programar la flash e indicarle que al acceder ignore los bytes de seguridad. De este modo nos permitirá acceder, con lo que podremos borrar la flash y programar el micro posteriormente.

Pero vayamos por partes. En la figura 3.1 se muestra el diálogo que aparece la primera vez que se intenta acceder al micro, en este caso particular, antes de acceder al simulador.

Podemos apreciar las distintas opciones mostradas, que a continuación pasamos a explicar:

- El tipo de conector que empleamos. Se refiere a las características del cable monitor empleado: si tiene control de la alimentación, reset, etc. En nuestro caso, la opción a elegir es Class I.
- La comunicación serie con el PC. Se indica el puerto y la velocidad. Normalmente 9600

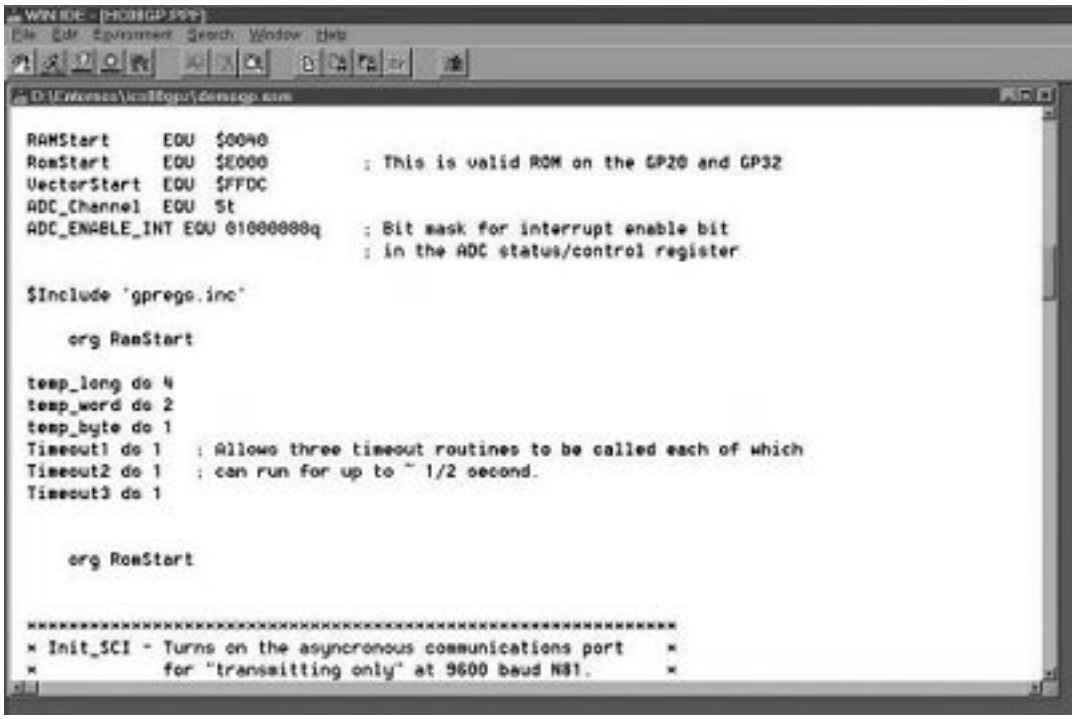

Figura 3.2: Entorno de edición y ensamblado

baudios.

- Los bytes de seguridad antes mencionados. Normalmente se emplea la primera opción, que es la que prueba con todas las combinaciones almacenadas de otros accesos. Como último recurso nos queda marcar la casilla IGNORE.
- El estado de la comunicación. En caso de no producirse un éxito en la comunicación podemos conocer en que parte se ha cometido el error.

Como detalle, en el caso de la simulación (al que pertenece la imagen anterior), podemos elegir la opción "SIMULATION only", que nos permite indicarle al simulador que solo vamos a simular el código, y que no necesitamos el micro.

Una vez hemos pasado con éxito este cuadro de diálogo, y dependiando de la clase del MON08 que tenemos, el entorno nos pedirá resetear el micro, cortarle la alimentación, etc (en nuestro caso, esto se realiza automáticamente).

3.5. Entorno de Edición y Ensamblado

En la figura 3.2 se muestra el entorno de edición.

Esta parte del entorno es la principal. En ella se pueden definir las características del resto de partes del entorno (menú Environment).

El primer paso es crear un projecto. Esto permitirá que se guarden todas las configuraciones de las distintas partes, así como el directorio de trabajo.

Lo principal en esta parte es editar el programa. Para ello, se puede abrir un fichero ya existente, o bien crear uno nuevo. Una vez se ha editado el fichero, es necesario ensamblarlo. Para ello basta con pulsar el primer icono de la barra de herramientas. A continuación aparecerá un cuadro donde se indica el nombre del fichero que se está ensamblado y el proceso de ensamblado (figura 3.3).

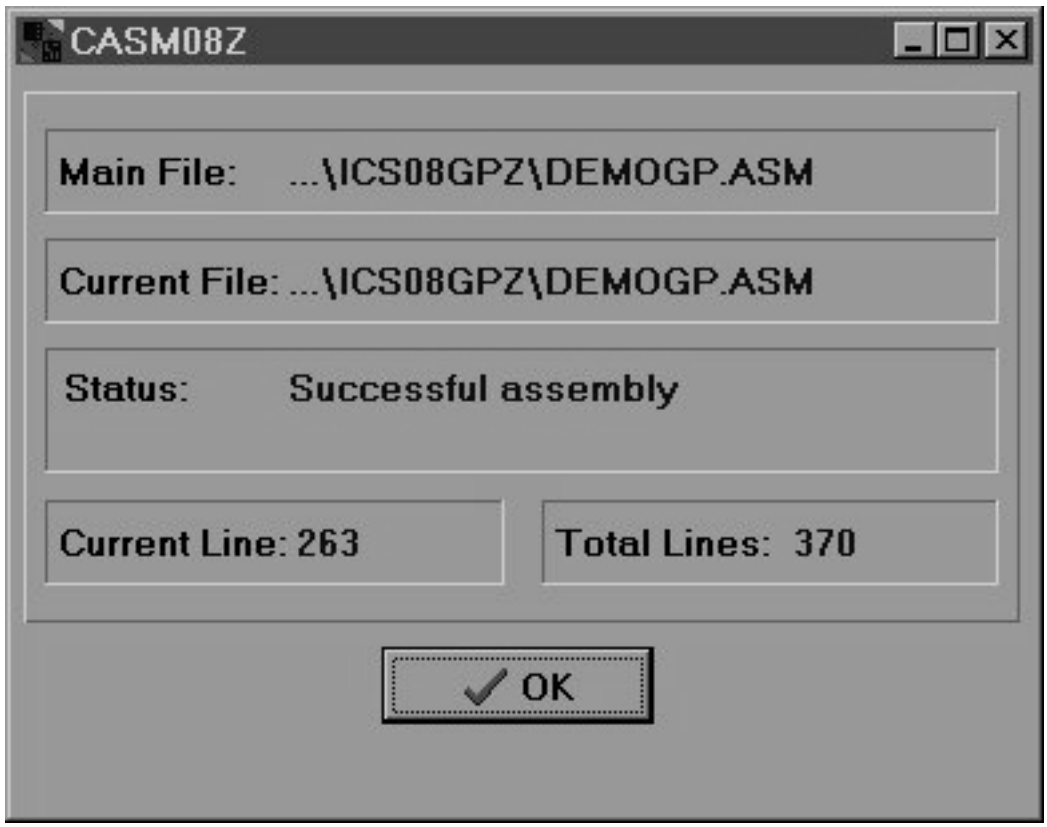

Figura 3.3: Ventana de compilación

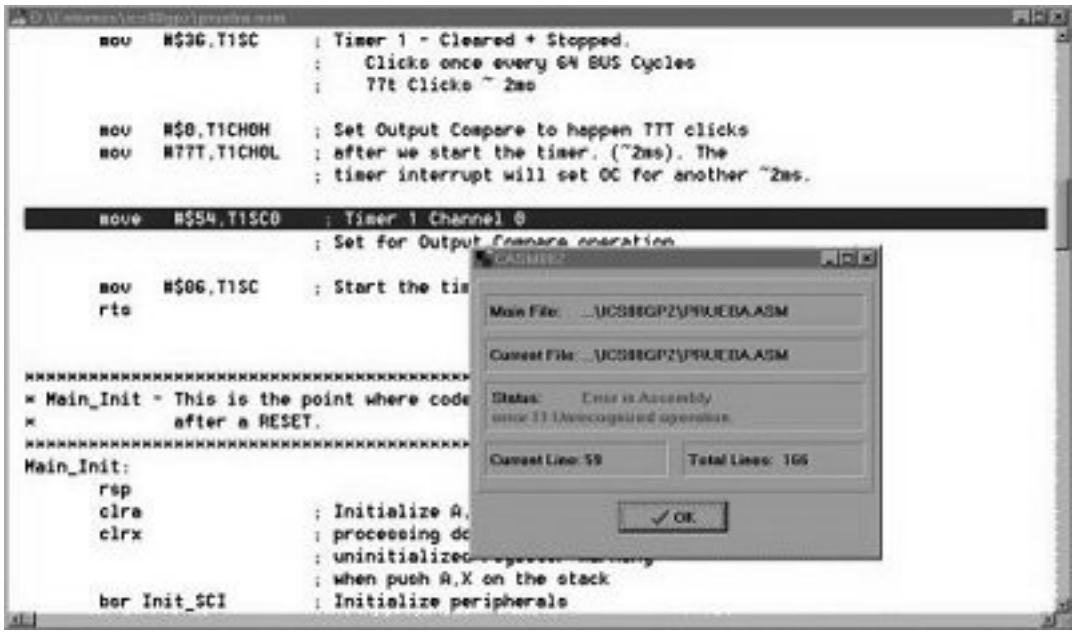

Figura 3.4: Ventana de compilación con error

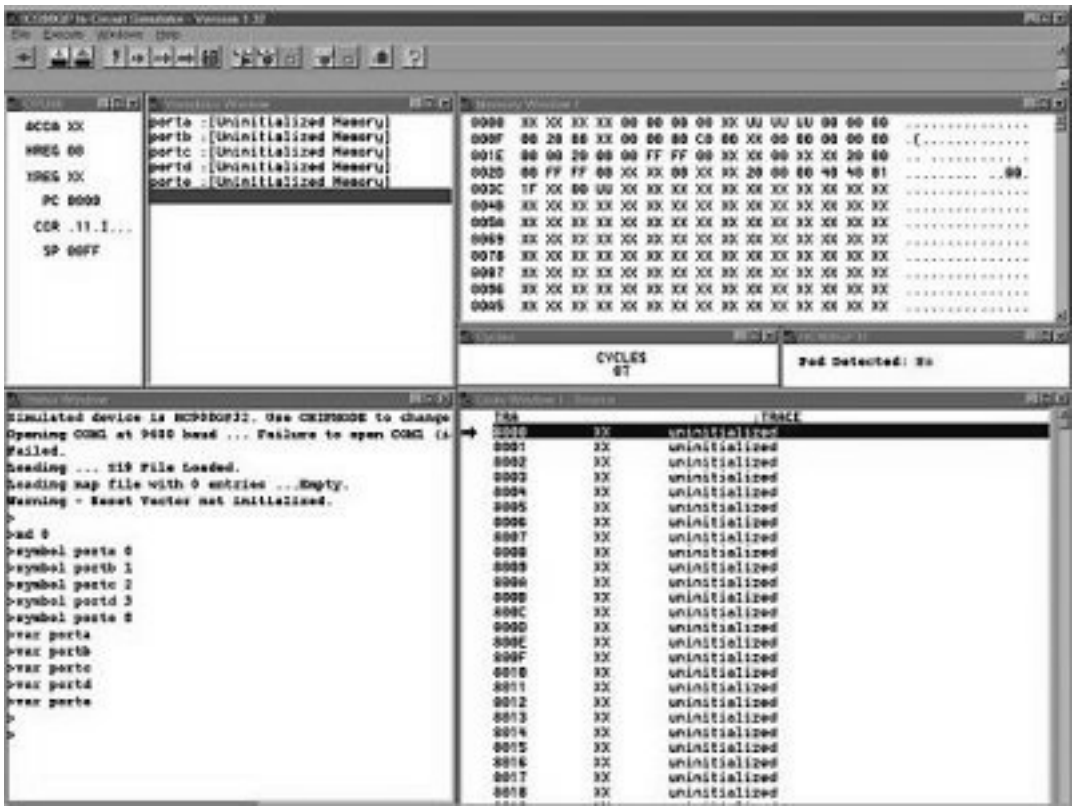

Figura 3.5: Entorno de simulación

En caso de que el código tenga errores (figura 3.4), estos aparecen tanto en el cuadro como en el editor (primero es necesario pulsar OK):

Inicialmente, si no existe ningún error, el cuadro desaparecerá al terminar el ensamblado. Sin embargo, existe la posibilidad de que éste se muestre siempre. Para ello, hay marcar en el menú Environment ->Setup Environment... ->Pestaña Assembler/Compiler ->la opción Wait for Assembler Result.

3.6. Entorno de Simulación

Antes de mostrarnos el entorno, el programa accede al microcontrolador a través del cable monitor, sin embargo, podemos acceder al entorno si no disponemos del micro, esperando a que nos de error y eligiendo la opción "SIMULATION only".

La simulación del micro se realiza por software en el ordenador, sin embargo, esta simulación tiene una importante característica, los valores de los puertos se toman directamente del micro. En otras palabras, se simula la ejecuación, pero los valores de entrada se corresponden con los reales, de modo que cualquier "periférico" conectado a algún puerto del micro se podrá manejar. Si se elige la opción "Simulation only" se pierde esta característica.

En la figura 3.5 se muestra el entorno de simulación.

Como puede apreciarse, está formado por un grupo de ventanas. Cada una de ellas tiene como misión mostrar contenido importante para llevar a cabo una correcta simulación del código:

- Valor de los registros del micro.
- Valor de los puertos. Además, permite mostrar cualquier variable del programa con más que introducir su nombre.

- Contenido de la memoria del micro.
- Número de ciclos de reloj ejecutados hasta el momento.
- Estado de la simulación.
- El código que se está simulando. Inicialmente éste se encuentra en ensamblador puro. Sin embargo es posible que aparezca el código tal y como se editó en el fichero, incluidos comentarios. Lo cual es muy práctico para no perderse.

El entorno de simulación dispone de varias opciones que nos permiten simular el código:

Estas opciones van desde la posibilidad de cargar el código fuente en la ventana de código, hasta ejecutar instrucción por instrucción el código. Además es posible poner breakpoints en el código.

A continuación pasamos a explicar algunas de ellas:

Permite cargar el código fuente en la ventana de programa. Esto es muy útil porque nos aparece el código tal y como lo hemos escrito, con los comentarios incluidos. El segundo botón permite recargar el programa.

Se trata de opciones de control de flujo del programa. La primera de ellas permite resetear la ejecución, volviendo al estado inicial. La segunda de ellas permite ejecutar una instrucción tras otra. La siguiente ejecuta las instrucciones una tras otra de forma consecutiva. La siguiente ejecuta el programa de forma continua, y la última, permite parar la ejecución.

Es posible añadir breakpoints en la ejecución. Para ello solo hay que señalar la línea donde se quiere insertar dentro de la ventana de programa y pulsando el botón derecho del ratón, seleccionar la opción Toggle Breakpoint at Cursor.

3.7. Entorno de Programación

Antes de mostrarnos el entorno, y tras comunicar con el micro, el programa nos pide el algoritmo a emplear para programar la flash del micro (figura 3.6).

Seleccionaremos 908_GP32.08P. Entonces nos aparece el entorno de programación (figura 3.7).

La parte fundamental de este entorno es la ventana de estado, en la que se muestra el resultado de los comandos ejecutados al pulsar alguno de los botones del entorno. Además permite introducir comandos manualmente.

Lo fundamental de este entorno es seleccionar el programa que se quiere grabar en la memoria flash mediante el icono "diskette". Una vez seleccionado el programa hay que borrar la memoria flash para eliminar el programa anterior. Esto puede hacerse mediante el icono "goma de borrar". Podemos además, chequear que realmente se encuentra borrada pulsando el icono "folio en blanco

Figura 3.6: Selección del algoritmo de programación

+ interrogación". Una vez borrada grabamos el programa mediante el icono "rayo", e igualmente podemos verificar que se ha grabado correctamente el programa, pulsando el icono a la derecha del icono "rayo".

El resultado de todas estas operaciones aparecerá en la ventana de estado.

3.8. Entorno de Debugger

Antes de mostrarnos el entorno, el programa accede al microcontrolador a través del cable monitor. La figura 3.8 nos muestra el aspecto del entorno de debug.

Como puede apreciarse, es idéntico al entorno de simulación, con la salvedad de que ha perdido algunas de las ventanas que aparecían en el entorno de simulación.

La funcionalidad de estas ventanas es la misma.

Lo único que hay que resaltar es que al contrario que la simulación In-Circuit, toda la ejecución se realiza en el micro.

Las opciones para debuggear el programa son identicas a las de la simulación, con la salvedad de que únicamente se puede poner un breakpoint. Esto se debe a que el debug se realiza en tiempo de ejecución del micro, y es muy complicado realizar un breakpoint hardware.

Por lo demás, es idéntico a la simulación.

3.9. Entorno de Simulación 2

Esta simulación tiene las mismas características que la simulación In-Circuit anterior, pero no se realiza acceso al micro. Por tanto, se trama de una simulación "estandard".

Figura 3.7: Entorno de programación

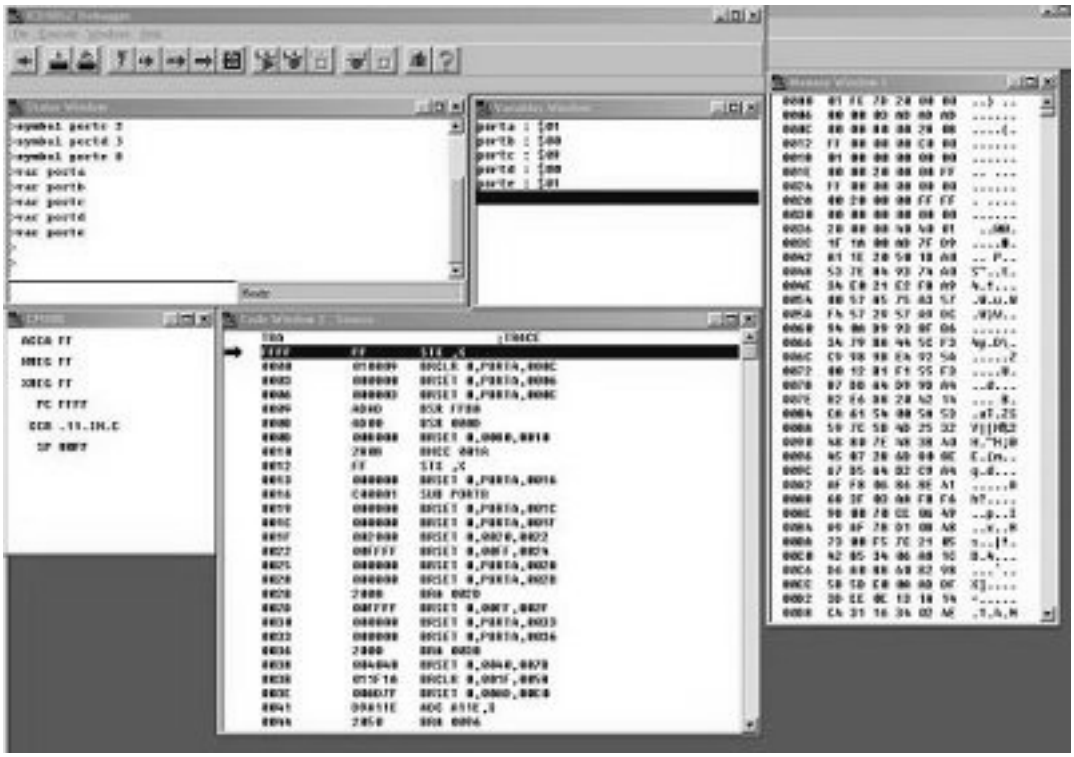

Figura 3.8: Entorno de Debug

Capítulo 4

Introducción al ensamblador del 68HC08

4.1. Antes de empezar...

Todo la información sobre el desarrollo de programas para el sistema GP_Bot se basa en el entorno de desarrollo ofrecido libremente por Motorola para la familia de microcontroladores MC68HC08. Este entorno emplea un ensamblador propio cuyas características no tienen porque corresponder con otros ensambladores para la misma familia de microcontroladores.

Aunque este documentos no pretende ser una ayuda a la programación en ensamblador, nos parece interesante hacer algunas observaciones.

4.2. Los registros

El microcontrolador 68HC08 dispone de un acumulador de 8 bits en el que se realizan todas las operaciona aritméticas y dos registros auxiliares también de 8 bits, el X y el H, que unidos se emplean para indexar posiciones de memoria de 16 bits. El uso del registro H es menor que el del registro X debido a que existen menor número de instrucciones que operan sobre él.

También se dispone de un puntero de pila, el cual debe apuntar a una posición de memoria RAM, y del registro de banderas de control (CCR).

4.3. Definición de las posiciones de memoria

Lo primero que hay que tener en cuenta antes de escribir un programa es establecer la división del contenido del mismo. En el caso de un microcontrolador, tenemos una parte que contiene las variables (RamStart), otra que contiene el programa y las constantes (RomStart) y otra con los vectores de interrupción (VectorStart). El inicio de cada una de estas partes dentro del esquema de memoria del microcontrolador, es el siguiente:

```
;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción

;;; VARIABLES DEL PROGRAMA
            ORG RamStart
```

```
;;; PROGRAMA
            ORG RomStart


;;; VECTORES
            ORG VectorStart
```

Los valores de inicio de cada unas de las posiciones de memoria se corresponden con las características del mapa de memoria del microcontrolador MC68HC908GP32.

La directiva `ORG` se emplea para indicarle al ensamblador que lo que aparece a continuación se debe situar a partir de la dirección de memoria indicada.

4.4. Diferencias entre dirección y valor

A la hora de programar en ensamblador es importante distinguir entre una dirección y un valor.

Una dirección viene dada por un número en hexadecimal, y por tanto, precedida del símbolo $. Si por ejemplo queremos leer el valor que está en la dirección $23F emplearemos `LDA $23F`.

Cuando nos referimos a un valor directamente debemos preceder al valor con el símbolo #. Con este símbolo indicamos que lo que viene a continuación es un valor inmediato. Existe distintas formas de indicar un valor inmediato:

- Si el valor se da en hexadecimal debemos precederlo del símbolo $. Por ejemplo `LDA #$0F`.
- Si el valor se da en decimal debemos poner detrás una T. Por ejemplo `LDA #15T`.
- Si el valor se da en binario debemos poner detrás una q. Por ejemplo `LDA #00001111q`.

Por defecto si a un valor no le acompaña ningún símbolo ($, T, q) se asume que viene dado en hexadecimal.

4.5. La definición de variables

Las variables se deben situar en memoria RAM, ya que este tipo de memoria es de lectura/escritura.

La definición de variables es la siguiente:

```
;;; VARIABLES DEL PROGRAMA
            ORG RamStart

variable1       DS 1      ;Reservamos un byte para la variable1
variable2       DS 3      ;Reservamos tres bytes para la variable2
```

Con la directiva DS reservamos el número de bytes que necesitemos para cada variable.

4.6. La definición de constante y programa

Las constantes deben definirse en la parte de la memoria que corresponde al programa, ya que como constantes que son, deben encontrarse en una memoria de solo lectura. En el caso de nuestro microcontrolador esta memoria es la FLASH.

```
;;; PROGRAMA
```

```
            ORG RomStart

;;; Constantes
constante1      FCB #11110000q
constante2      FCB #$F0
constante3      FCB 10T,15T
```

Como puede comprobarse, una constante tiene uno más bytes asignados desde su declaración y no es posible cambiar su valor.

La declaración de constantes se debe hacer antes de comenzar con el programa.

```
;;; PROGRAMA
            ORG RomStart
;;; Constantes
constante1      FCB #11110000q
constante2      FCB #$F0
constante3      FCB 10T,15T

;;; Programa
main:
            JMP main
```

El programa debe comenzar siempre con la declaración de una etiqueta de comienzo que defina donde comienza el programa para posteriormente indicarselo a la CPU haciendo que el vector de reset apunté a esa etiqueta (ver 4.8).

Es importante que el programa termine con un bucle que le envíe al inicio del programa, aunque no tiene porque llegar en ningún momento a ejecutar esta instrucción pero evitará que en algún momento se ejecuten instrucciones que no pertenecen al programa.

4.7. La pila

Una de las partes más importantes de un programa es el establecimiento de la pila. La pila es una estructura de datos dinámica y por tanto debe establecerse en posiciones de memoria correspondientes a la memoria RAM. Esto implica que es muy importante controlar el tamaño de la RAM empleada para variables y para la pila con el fin de que la pila no alcance posiciones de memoria donde residan variables. El microcontrolador MC68HC908GP32 dispone de 512 bytes de RAM, lo cual implica que si la RAM comienza en la dirección $0040 debe terminar en la $23F. Situaremos la pila al final de la memoria RAM, es decir, en la dirección $23F. Para ello empleremos la siguiente secuencia de instrucciones:

```
            ;Inicializamos pila
            LDHX #InitStk
            TXS
```

Hasta ahora la estructura del programa tiene el siguiente aspecto:

```
;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción
InitStk     EQU $23F      ;Inicio de la pila

;;; VARIABLES DEL PROGRAMA
```

```
            ORG RamStart


;;; PROGRAMA
            ORG RomStart
main:
            ;Inicializamos pila
            LDHX #InitStk
            TXS

            JMP main


;;; VECTORES
            ORG VectorStart
```

4.8. Los vectores de interrupción

En la zona de memoria correspondiente a los vectores de interrupción es necesario indicar la dirección a la que debe de saltar la ejecución del programa en caso de que alguna de las interrupciones ocurra y esta se encuentre habilitada.

Inicialmente un programa no tiene porque emplear interrupciones. En este caso es igualmente necesario dar una dirección de referencia a los vectores de interrupción. El único vector de interrupción que siempre debe tener una dirección correctamente asignada es el vector de reset. Los demás simplemente hay que dirigirlos hacía una dirección que les mande retornar como medida de precaución.

```
;;; VECTORES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DUMMY-ISR - Rutina de Servicio de Interrupcion "que no hace nada".
;;; Simplemente retornamos.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dummy_isr:
            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Vectores de interrupción.
;;; Indicamos las rutinas de servicio para cada interrupción.
;;; En caso de no emplear alguna, apuntar a la rutina dummy_isr.
;;; El Vector de Reset debe apuntar al inicio del programa.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ORG VectorStart
            DW dummy_isr     ;Vector TMB
            DW dummy_isr     ;Vector DAC
            DW dummy_isr     ;Vector KBI
            DW dummy_isr     ;Vector Transmisor SCI
            DW dummy_isr     ;Vector Receptor SCI
            DW dummy_isr     ;Vector Error SCI
            DW dummy_isr     ;Vector Transmisor SPI
            DW dummy_isr     ;Vector Receptor SPI
            DW dummy_isr     ;Vector Overflow TIM2
```

```
DW dummy_isr     ;Vector Canal 1 TIM2
DW dummy_isr     ;Vector Canal 0 TIM2
DW dummy_isr     ;Vector Overflow TIM1
DW dummy_isr     ;Vector Canal 1 TIM1
DW dummy_isr     ;Vector Canal 0 TIM1
DW dummy_isr     ;Vector PLL
DW dummy_isr     ;Vector IRQ1
DW dummy_isr     ;Vector SWI
DW main          ;Vector Reset
```

4.9. El juego de instrucciones

A continuación presentamos una breve descripción de las instrucciones de la familia 68HC08 agrupadas según su funcionalidad. Para más detalles consultar el documento técnico sobre la familia 68HC08 disponible en la web de Motorola.

4.9.1. Instrucciones de carga, almacenamiento y transferencia

CARGA: Estas instrucciones permiten cargan valores en los registros , leer de memoria, de los puertos, etc.

LDA: Permite cargar un valor de 8 bits en el acumulador.

LDHX: Permite cargar un valor de 8 o 16 bits en los registros H y X.

LDX: Permite cargar un valor de 8 bits en el registro X.

CLRA: Pone a cero el acumulador.

CLRH: Pone a cero el registro H.

CLRX: Pone a cero el registro X.

RSP: Resetea el SP (Puntero de Pila) a $FF.

ALMACENAMIENTO: Estas instrucciones permiten alterar una posición de memoria, registros internos, puertos, etc.

STA: Permite almacenar el contenido del acumulador en una posición de memoria (8 bits).

STHX: Permite almacenar el contenido de H:X en una posición de memoria (16 bits).

STX: Permite almacenar el contenido del registro X en una posición de memoria (8 bits).

CLR: Pone a cero una posición de memoria.

MOV: Almacenar un valor en una posición de memoria.

TRANSFERENCIA: Estas instrucciones permiten transferir el contenido entre registros, entre registros y memoria o entre dos posiciones de memoria.

PSHA: Permite transferir el acumulador a la pila.

PSHH: Permite transferir el registro H a la pila.

PSHX: Permite transferir el registro X a la pila.

PULA: Permite transferir el contenido de la pila al acumulador.

PULH: Permite transferir el contenido de la pila al registro H.

PULX: Permite transferir el contenido de la pila al registro X.

TAP: Permite transferir el acumulador a CCR (Registro de Códigos de Condición).

TAX: Permite transferir el acumulador al registro X.

TPA: Permite transferir el CCR al acumulador.

TSX: Permite transferir SP al registro X.

TXA: Permite transferir el registro X al acumulador.

TXS: Permite transferir el registro X al SP (el contenido de H y X).

MOV: Permite transferir el contenido de una posición de memoria a otra posición de memoria.

4.9.2. Intrucciones aritméticas

SUMAR:

ADC: Sumar al acumulador un dato con accareo.

ADD: Sumar al acumulador un dato sin accareo.

AIS: Sumar un valor inmediato al SP (con signo).

AIX: Sumar un valor inmediato a H:X (con signo).

INC: Incrementar el valor de una posición de memoria.

INCA: Incrementar el acumulador.

INCX: Incrementar el valor del registro X.

RESTAR:

SBC: Restar al acumulador un dato sin accareo.

SUB: Restar al acumulador un dato.

DEC: Decrementar el valor de una posición de memoria.

DECA: Decrementar el acumulador.

DECX: Decrementar el valor del registro X.

MULTIPLICAR:

MUL: Multiplicar el valor del registro X por el acumulador (sin signo).

DIVIDIR:

DIV: Dividir el valor de los registros H:A entre el valor del registro X.

COMPARACIONES:

CMP: Compara el acumulador con un valor inmediato o posición de memoria.

CPX: Compara el contenido del registro X con un valor inmediato o posición de memoria.

COMPLEMENTO-A-DOS: Obtención de número negativos.

NEG: Complemento a dos de una posición de memoria.

NEGA: Complemento a dos del acumulador.

NEGX: Complemento a dos del valor del registro X.

4.9.3. Operaciones lógicas y manipulación de bits

AND: Operación AND lógico entre el acumulador y un valor inmediato o posición de memoria.

EOR: Operación XOR lógico entre el acumulador y un valor inmediato o posición de memoria.

ORA: Operación OR lógico entre el acumulador y un valor inmediato o posición de memoria.

BCRL: Pone a cero un bit determinado de una posición de memoria.

BIT: Se realiza un AND lógico entre el acumulador y una posición de memoria pero sin alterar ninguno de ellos. El resultado se refleja en el bit Z del registro CCR.

BSET: Pone a uno un bit determinado de una posición de memoria.

CLR: Pone a cero una posición de memoria.

CLRA: Pone a cero el acumulador.

CLRX: Pone a cero el registro X.

CLRH: Pone a cero el registro H.

COM: Complemento a uno de una posición de memoria.

COMA: Complemento a uno del acumulador.

COMX: Complemento a uno del registro X.

DAA: Ajuste decimal del acumulador.

NSA: Intercambia los cuatro bits menos significativos con los cuatro bits más significativos del acumulador.

TST: Comprueba si una posición de memoria es negativa o cero. Afecta a los bits Z y N del CCR.

TSTA: Comprueba si el acumulador es negativo o cero. Afecta a los bits Z y N del CCR.

TSTX: Comprueba si el contenido del registro X es negativo o cero. Afecta a los bits Zy N del CCR.

4.9.4. Desplazamientos y rotaciones

ASL: Igual que LSL. Rotación aritmética a la izquierda de una posición de la memoria.

ASLA: Igual que LSLA. Rotación aritmética a la izquierda del acumulador.

ASLX: Igual que LSLX. Rotación aritmética a la izquierda del registro X.

ASR: Rotación aritmética a la derecha de una posicion de memoria.

ASRA: Rotación aritmética a la derecha del acumulador.

ASRX: Rotacion aritmética a la derecha del registro X.

LSL: Igual que ASL. Rotación lógica a la izquierda de una posicion de memoria.

LSLA: Igual ASLA. Rotación lógica a la izquierda del acumulador.

LSLX: Igual ASLX. Rotación lógica a la izquierda del registro X.

LSR: Rotación lógica a la derecha de una posicion de memoria.

LSRA: Rotación lógica a la derecha del acumulador.

LSRX: Rotación lógica a la derecha del registro X.

ROL: Rotación con acarreo a la izquierda de una posicion de memoria.

ROLA: Rotación con acarreo a la izquierda del acumulador.

ROLX: Rotación con acarreo a la izquierda del registro X.

ROR: Rotación con acarreo a la derecha de una posición de memoria.

RORA: Rotación con acarreo a la derecha del acumulador.

RORX: Rotación con acarreo a la derecha del registro X.

4.9.5. Bifurcaciones y saltos

BCC: Bifurcar si acarreo está a cero.

BCS: Bifurcar si acarreo está a uno. Igual que BLO.

BEQ: Bifurcar si igual (Z=1).

BGE: Bifurcar si mayor o igual (con signo).

BGT: Bifurcar si mayor (con signo).

BHCC: Bifurcar si acarreo-medio está a cero.

BHCS: Bifurcar si acarreo-medio está a uno.

BHI: Bifurcar si mayor (sin signo).

BHS: Bifurcar si mayor o igual (sin signo). Igual que BCC.

BIH: Bifurcar si el pin IRQ está a uno.

BIL: Bifurcar si el pin IRQ está a cero.

BLE: Bifurcar si menor o igual (con signo).

BLO: Bifurcar si menor (sin signo). Igual a BCS.

BLS: Bifurcar si menor o igual (sin signo).

BLT: Bifurcar si menor (con signo).

BMC: Bifurcar si el bit de mśscara de interrupcion está a cero.

BMI: Bifurcar si negativo (N=1).

BMS: Bifurcar si el bir de máscara de interrupción esta a uno.

BNE: Bifurcar si no igual (Z=0).

BPL: Bifurcar si positivo (N=0).

BRA: Bifurcar (salto incondicional).

BRCLR: Bifurcar si un bit determinado de una posición de memoria está a cero.

BRN: Nunca bifurcar ¿?

BRSET: Bifurcar si un bit determinado de una posición de memoria está a uno.

BSR: Bifurcar a una subrutina (salto incondicional).

CBEQ: Comparar acumulador con una posición de memoria y bifurcar si igual.

CBEQA: Comparar el acumulador con un valor inmediato y bifurcar si igual.

CBEQX: Comparar el registro X con un valor inmediato y bifurcar si igual.

DBNZ: Decrementar una posición de memoria y bifurcar si no es cero.

DBNZA: Decrementar el acumulador y bifurcar si no es cero.

DBNZX: Decrementar el registro X y bifurcar si no es cero.

JMP: Salto incondicional.

JSR: Salto incondicional a una subrutina.

4.9.6. Instrucciones de modificación de los bits del registro CCR

CLC: Poner a cero el bit de accareo.

CLI: Poner a cero el bit de máscara de interrupción.

SEC: Poner a uno el bit de accareo.

SEI: Poner a uno el bit de máscara de interrupción.

TAP: Mover el acumulador al CCR.

TPA: Mover el CCR al acumulador.

4.9.7. Otras instrucciones

NOP: No hace nada. Consume un ciclo de reloj.

RTI: Retornar de una interrupción.

RTS: Retornar de una subrutina.

STOP: Parar el reloj.

SWI: Interrupción software.

Capítulo 5

Programación en la tarjeta GP_Bot

A continuación se muestran ejemplos de programación de los distintos recursos que ofrece el sistema de desarrollo GP_Bot, así como aquellos sensores para los que está especialmente diseñada la placa GP_Bot_Ifaz. Es imprescindible haber leido antes el manual de usuario del sistema de desarrollo GP_Bot para tener una idea exacta de donde se encuentra cada una de las conexiones o pines que se mencionan en los ejemplos.

Una operación que nos va a facilitar la programación es añadir las definiciones de los registros del microcontrolador. Estas definiciones asocian nombres facilmente reconocibles a las direcciones de los registros. Las definiciones que se emplearán durante el resto del documento se corresponde con las existentes en el fichero 'gpregs.inc' cuya descripción podemos encontrar en 6.1.

5.1. El COP

El COP es un sistema de WatchDog hardware que llevan los microcontroladores para evitar situaciones de "cuelgue" del programa.

Su operación se basa en decrementar un contador de modo que si este contador llega a cero el microcontrolador se resetea. Esto permite evitar que un programa caiga en un bucle infinito si durante la ejecución correcta del programa, en puntos determinados, se escribe el contador, evitando que llegue a cero salvo que la ejecución del programa no pase por los puntos del programa donde se ha decidido escribir el contador.

Este sistema es muy importante en aplicaciones críticas. En nuestro caso, se podría emplear para evitar que el robot entrase en una etapa de ejecución indeterminada. Sin embargo, lo vamos a desactivar al inicio del programa para evitar que el microcontrolador se resetee durante la ejecución de los programas. Si se desea emplearlo o saber más sobre el COP se recomienda leer la Sección 9 del documento técnico sobre el MC68HC908GP32 que está disponible en la web de Motorola.

A continuación se presenta el código necesario para deshabilitar el COP:

```
   ;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción
InitStk     EQU $23F      ;Inicio de la pila

;;; VARIABLES DEL PROGRAMA
             ORG RamStart


;;; PROGRAMA
```

```
            ORG RomStart

main:
            ;Inicializamos pila
            LDHX #InitStk
            TXS

            ;Deshabilitamos el COP
            BSET 0,CONFIG1

            JMP main


;;; VECTORES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DUMMY-ISR - Rutina de Servicio de Interrupcion "que no hace nada"
;;; Simplemente retornamos
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dummy_isr:
            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Vectores de interrupción.
;;; Indicamos las rutinas de servicio para cada interrupción.
;;; En caso de no emplear alguna, apuntar a la rutina dummy_isr.
;;; El Vector de Reset debe apuntar al inicio del programa.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ORG VectorStart
            DW dummy_isr     ;Vector TMB
            DW dummy_isr     ;Vector DAC
            DW dummy_isr     ;Vector KBI
            DW dummy_isr     ;Vector Transmisor SCI
            DW dummy_isr     ;Vector Receptor SCI
            DW dummy_isr     ;Vector Error SCI
            DW dummy_isr     ;Vector Transmisor SPI
            DW dummy_isr     ;Vector Receptor SPI
            DW dummy_isr     ;Vector Overflow TIM2
            DW dummy_isr     ;Vector Canal 1 TIM2
            DW dummy_isr     ;Vector Canal 0 TIM2
            DW dummy_isr     ;Vector Overflow TIM1
            DW dummy_isr     ;Vector Canal 1 TIM1
            DW dummy_isr     ;Vector Canal 0 TIM1
            DW dummy_isr     ;Vector PLL
            DW dummy_isr     ;Vector IRQ1
            DW dummy_isr     ;Vector SWI
            DW main          ;Vector Reset
```

Al final nos ha resultado un esqueleto de programa a partir del cual podemos comenzar a desarrollar nuestros programas.

5.2. La tarjeta GP_Bot_Ifaz

El sistema GP_Bot fue diseñado para proveer de todos los recursos necesarios para el desarrollo de aplicaciones de robótica autónoma. Dentro de este sistema, la tarjeta GP_Bot_Ifaz se encarga de poner a disposición del desarrollador una serie de conectores especiales para conectar sensores o dispositivos espécificos.

Teniendo esto en mente, es importante conocer la relación entre los conectores de la tarjeta GP_Bot_Ifaz y los puertos de la placa GP_Bot. Por tanto, antes de programar cualquier dispositivo o sensor es importante conocer en que puertos se encuentra conectado.

Llegados a este punto sería importante echar un vistazo al manual de usuario del sistema de desarrollo GP_Bot.

5.3. Programación de motores CC

La tarjeta GP_Bot_Ifaz viene preparada para el control de cuatro motores CC o dos motores PAP. Esto se debe a que dispone de dos drivers L293 cada uno de los cuales se encuentra capacitado para controlar dos motores CC o un motor PAP. El uso de este IC simplifica el control a 2 bits por motor siguiendo el siguiente esquema:

Bit 2	**Bit 1**	**Operación**
0	0	No se mueve
0	1	Movimiento sentido A
1	0	Movimiento sentido B
1	1	No permitido

El sentido del movimiento depende de la asociación de los cables del motor con los conectores. Si por ejemplo con la combinación 01 el motor se mueve hacia adelante, con la combinación 10 el motor irá hacia atrás.

Una vez conocemos el funcionamiento del IC que controla los motores, debemos decidir donde conectar los motores. La tarjeta GP_Bot_Ifaz nos ofrece cuatro conectores que están asociados a un grupo de pines de los puertos C y A. Existe por tanto una asociación entre los pines de control, los ICs L293 y los cuatro conectores para motores.

Puerto	**Bit 2**	**Bit 1**	**Conector GP_Bot_Ifaz**
A	4	3	J15
A	2	1	J14
C	4	3	J13
C	2	1	J12

A continuación se muestra un sencillo programa que mueve dos motores conectados en J15 y J14. Es importante determinar la combinación de bits que hay que poner en los drivers para que los motores se muevan en el sentido esperado. En nuestro caso, la combinación 01 mueve los motores hacia adelante y la combinación 10 los mueve hacia atrás. Puede resultar que una misma combinación mueva un motor en un sentido y al otro motor en el sentido contrario. La combinación a emplear depende de la conexión de los cables del motor en el conector.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; CLUB ROBOTICA - MECATRONICA   ETSI - UAM
;;;
;;; SISTEMA DE DESARROLLO GP_Bot
;;;
;;; Programa ejemplo de manejo de motores CC.
;;;
```

```
;;; Queremos manejar dos motores de CC situados en los conectores J14 y J15.
;;; Es muy importante conocer los valores que determinan el movimiento de cada
;;; motor, lo cual depende de la conexión de los cables de alimentación.
;;; No ha sido necesario definir variables.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción
InitStk     EQU $23F      ;Inicio de la pila

;;;;; Incluimos las definiciones de los registros del micro
$Include 'gpregs.inc'

;;; DEFINICION DEL MOVIMIENTO DEL MOTOR
;;; Motor derecho en pines PTA2 y PTA1 (J14)
;;; Motor izquierdo en pines PTA4 y PTA3 (J15)
;;; 01 hacia adelante
;;; 10 hacia atras
ADELANTE    EQU 00001010q ;Ambos motores adelante
ATRAS       EQU 00010100q ;Ambos motores atras
DERECHA     EQU 00001100q ;Derecho atras / izquierdo adelante
IZQUIERDA   EQU 00010010q ;Derecho adelante / izquierdo atras

;;; VARIABLES DEL PROGRAMA
            ORG RamStart


;;; PROGRAMA
            ORG RomStart

main:
            ;Inicializamos pila
            LDHX #InitStk
            TXS

            ;Deshabilitamos el COP
            BSET 0,CONFIG1

            ;Configuramos los pines PTA4, PTA3, PTA2 y PTA1 como salida
            ;El resto de pines no importan (los configuramos como salidas
            ;también)
            LDA #$FF
            STA DDRA

            ;Inicio de la secuencia de movimiento
secuencia:
            ;Movemos el robot hacia adelante
            LDA #ADELANTE
            STA PORTA
            ;Retardo proporcional al desplazamiento deseado
```

```
            ;(depende de los motores del robot)

            ;Movemos el robot hacia la derecha
            LDA #DERECHA
            STA PORTA
            ;Retardo proporcional al desplazamiento deseado
            ;(depende de los motores del robot)

            ;Movemos el robot hacia atras
            LDA #ATRAS
            STA PORTA
            ;Retardo proporcional al desplazamiento deseado
            ;(depende de los motores del robot)

            ;Movemos el robot hacia la izquierda
            LDA #IZQUIERDA
            STA PORTA
            ;Retardo proporcional al desplazamiento deseado
            ;(depende de los motores del robot).

            ;Repetimos la secuencia
            JMP secuencia


;;; VECTORES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DUMMY-ISR - Rutina de Servicio de Interrupcion "que no hace nada"
;;; Simplemente retornamos
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dummy_isr:
            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Vectores de interrupción.
;;; Indicamos las rutinas de servicio para cada interrupción.
;;; En caso de no emplear alguna, apuntar a la rutina dummy_isr.
;;; El Vector de Reset debe apuntar al inicio del programa.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ORG VectorStart
            DW dummy_isr     ;Vector TMB
            DW dummy_isr     ;Vector DAC
            DW dummy_isr     ;Vector KBI
            DW dummy_isr     ;Vector Transmisor SCI
            DW dummy_isr     ;Vector Receptor SCI
            DW dummy_isr     ;Vector Error SCI
            DW dummy_isr     ;Vector Transmisor SPI
            DW dummy_isr     ;Vector Receptor SPI
            DW dummy_isr     ;Vector Overflow TIM2
            DW dummy_isr     ;Vector Canal 1 TIM2
            DW dummy_isr     ;Vector Canal 0 TIM2
            DW dummy_isr     ;Vector Overflow TIM1
            DW dummy_isr     ;Vector Canal 1 TIM1
            DW dummy_isr     ;Vector Canal 0 TIM1
```

```
            DW dummy_isr        ;Vector PLL
            DW dummy_isr        ;Vector IRQ1
            DW dummy_isr        ;Vector SWI
            DW main             ;Vector Reset
```

5.4. Programación de los sensores de infrarrojos CNY70

La tarjeta GP_Bot_Ifaz dispone de cuatro conectores especialmente preparados para los sensores CNY70, identificados como J8, J9, J10 y J11. Estos conectores presentan cuatro pines que aportan Vcc, GND, una entrada digital y una entrada analógica para cada sensor.

IMPORTANTE: El pin de VCC de estos conectores no se corresponde con +5 V sino que se encuentra regulado por una resistencia que es necesaria para el sensor CNY70. Por tanto, es posible que conectando a este pin otros sensores no funcionen adecuadamente.

Las entradas analógicas asociadas a estos conectores se corresponden con pines del puerto B, dado que es este puerto el que comparte sus pines con el conversor A/D. De igual modo, las entradas digitales se corresponden con un par de pines de los puertos A y C.

Conector	E. Analógica	E. Digital	GND	VccR
J8	PTB0	PTC5	*	*
J9	PTB1	PTC6	*	*
J10	PTB2	PTA5	*	*
J11	PTB3	PTA6	*	*

Si se desarrolla adecuadamente el montaje del sensor, solo es necesario elegir el conector y conocer que pines se encuentran asociados a ese conector.

A continuación se muestra un ejemplo de lectura del sensor CNY70 tanto en valor digital como analógico.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; CLUB ROBOTICA - MECATRONICA   ETSI - UAM
;;;
;;; SISTEMA DE DESARROLLO GP_Bot
;;;
;;; Programa ejemplo de como leer el resultado de un sensor CNY70 mediante
;;; una entrada digital y una entrada analogica.
;;; Se emplea el conversor A/D para digitalizar la entrada analogica.
;;;
;;; Queremos leer informacion del sensor CNY70 situado en el conector J8.
;;; Se emplean el pin PTC5 para entrada digital y PTB0 para la analogica.
;;;
;;; Los valores leidos se almacenan en las variables correspondientes:
;;;     - v_dig : variable con el valor digital obtenido
;;;     - v_ad : variable con el valor analogico convertido a digital
;;;
;;; Las resistencias de pull-up del puerto B son opcionales.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción
```

```
InitStk     EQU $23F      ;Inicio de la pila

;;; Incluimos las definiciones de los registros del micro
$Include 'gpregs.inc'

;;; VARIABLES
            ORG RamStart
v_dig DS 1              ;variable para valor digital
v_ad  DS 1              ;variable para valor analogico convertido


;;; PROGRAMA
            ORG RomStart

main:
            ;Inicializamos pila
            LDHX #InitStk
            TXS

            ;Deshabilitamos el COP
            BSET 0,CONFIG1

            ;CONFIGURAR PIN DE ENTRADA DIGITAL
            ;Sensor en pin PTC5 para entrada digital
            ;Configurar como entrada PTC5 (poner a 0)
            BCLR 5,DDRC     ;El bit 5 = 0, el resto como estan

            ;Activar pull-up del PTC5 (poner a 1)
            BSET 5,PTCPUE     ;El bit 5 = 1, el resto como estan

            ;CONFIGURAR CONVERSOR A/D
            ;Configuramos reloj - Registro ADCLK
            ;Vamos a emplear el reloj del bus interno ->bit 4 = 1
            ;Los bits 7 a 5 se emplean para manipular la señal de reloj.
            ;Como la frecuencia del bus interno es ~2,45 MHz y la documentacion
            ;dice que el reloj del conversor debe ser ~1 MHz, debemos dividirla
            ;entre 2 ->bit 7 = 0, bit 6 = 0 y bit 5 = 1
            ;Resto de bits no se emplean (poner cualquier valor)
            LDA #00110000q
            STA ADCLK

bucle:
            ;Inicializamos las variables a 0
            CLRA
            STA v_dig
            STA v_ad

            ;La entrada analogica se realiza por el bit PTB0, o lo que es lo
            ;mismo el canal AD0. Por tanto debemos activar este canal.
            ;El registro a manejar es ADSCR.
            ;Como queremos que se realize una unica conversion, tenemos que
            ;configurar el modo no continuo ->bit 5 = 0
            ;No vamos a emplear la interrupcion ->bit 6 = 0
            ;El bit 7 no se configura (ponemos cualquier valor)
```

```
            ;Los bits 4 a 0 se emplean para seleccionar el canal. Seleccionamos
            ;el AD0 ->bist 4 a 0 = 0
            LDA ADSCR
            AND #10000000q
            STA ADSCR

            ;El bit mas significativo del registro ADSCR nos indica cuando termina
            ;la conversion poniendose a 1. Esperamos a que este a 1.
sigue:
            BRCLR 7,ADSCR,sigue

            ;El resultado de la conversion se encuentra en el registro ADR.
            ;Leemos el resultado de la conversion y lo guardamos en la variable
            ;correspondiente
            LDA ADR
            STA v_ad

            ;Leemos el pin PTC5 con el resultado digital, y lo guardamos en la
            ;variable correspondiente
            LDA PORTC
            AND #$20     ;Eliminamos el resto de valores del puerto
            LSLA         ;Desplazamiento logico del contenido de A tres
            LSLA         ;posiciones a la izq. con el objetivo de que el bit
            LSLA         ;leido se guarde en el flag de carry (flag C)
            ROL v_dig    ;Rotamos a la izquierda la variable, y el bit de carry
                         ;se almacena en el bit menos significativo de la variable

            JMP bucle

;;; VECTORES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DUMMY-ISR - Rutina de Servicio de Interrupcion "que no hace nada"
;;; Simplemente retornamos
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dummy_isr:
            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Vectores de interrupción.
;;; Indicamos las rutinas de servicio para cada interrupción.
;;; En caso de no emplear alguna, apuntar a la rutina dummy_isr.
;;; El Vector de Reset debe apuntar al inicio del programa.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ORG VectorStart
            DW dummy_isr     ;Vector TMB
            DW dummy_isr     ;Vector DAC
            DW dummy_isr     ;Vector KBI
            DW dummy_isr     ;Vector Transmisor SCI
            DW dummy_isr     ;Vector Receptor SCI
            DW dummy_isr     ;Vector Error SCI
            DW dummy_isr     ;Vector Transmisor SPI
            DW dummy_isr     ;Vector Receptor SPI
            DW dummy_isr     ;Vector Overflow TIM2
```

```
DW dummy_isr     ;Vector Canal 1 TIM2
DW dummy_isr     ;Vector Canal 0 TIM2
DW dummy_isr     ;Vector Overflow TIM1
DW dummy_isr     ;Vector Canal 1 TIM1
DW dummy_isr     ;Vector Canal 0 TIM1
DW dummy_isr     ;Vector PLL
DW dummy_isr     ;Vector IRQ1
DW dummy_isr     ;Vector SWI
DW main          ;Vector Reset
```

5.5. Control de velocidad de motores CC mediante PWM

El manejo de motores CC no solo pasa por hacer que estos se muevan hacia adelante o hacia atrás en función de nuestra necesidad, sino que en ocasiones es necesario poder controlar la velocidad de los mismos. Debido a que los motores CC no disponen de ningún cable de control que nos permita controlar su velocidad, es necesario jugar con la alimentación del motor.

El sistema que se propone en este ejemplo consiste en controlar el tiempo que los motores se encuentran alimentados, de modo que durante un cierto tiempo se alimentan y durante otro cierto tiempo no se alimentan. Este efecto de alimentar y cortar la alimentación equivale a dividir la cantidad de potencia que reciben los motores.

5.5.1. Control de Potencia mediante una señal PWM

La idea de este sistema es tomar una unidad de tiempo T, no demasiado grande, por ejemplo de microsegundos. Esta unidad de tiempo será la unidad de tiempo que alimentaremos al motor. Si el motor se encuentra alimentado durante esta unidad de tiempo, infinitas unidades de tiempo resultan en una alimentación continua del motor, o lo que es lo mismo, el motor estará todo el tiempo a máxima velocidad (ver figura 5.2).

Supongamos por un momento que durante la mitad de esta unidad de tiempo no alimentamos el motor, es decir, si durante T/2 el motor es alimentado y durante otros T/2 el motor no es alimentado. El efecto que conseguimos es reducir la máxima velocidad del motor a la mitad (ver figura 5.1), y además, como la unidad de tiempo T que hemos elegido es demasiado pequeña, el efecto de aceleración-parada no es perceptible por nosotros, pero el motor si que lo sufre. Supongamos ahora que en vez de alimentar el motor T/2 lo hacemos solo durante T/4, entonces habremos conseguido que el motor vaya a la cuarta parte de velocidad máxima que tiene. Como se aprecia, el sistema tiene multiples posibilidades.

5.5.2. Implementación en el microcontrolador

La implementación de este controlador de potencia se puede realizar de varias maneras. Sin embargo, quizás la más cómoda sea empleando un temporizador y sus interrupciones asociadas (ver figura 5.1). El microcontrolador que estamos manejando dispone de dos timers o temporizadores (ver sección 2.2). Para esta aplicación solo será necesario emplear uno de ellos.

Supongamos que escogemos una unidad de tiempo de alimentación T tal que T = X*Y donde X es el periodo de reloj seleccionado e Y es el número de ciclos que queremos contar. En el caso del ejemplo, se ha tomado un periodo de reloj X = 6,5 useg y un número de ciclos 30, tal que T = 6,5 useg * 30 = 195 useg. Configuramos el temporizador para que se incremente cada X tiempo y establecemos el valor máximo que va a contar, Y. Cuando el temporizador llega al valor establecido Y, o lo que es lo mismo, ha pasado un tiempo X*Y, se provoca la interrupción de Overflow. Esta interrupcion nos indica el final de una unidad de alimentación o lo que es lo mismo, el inicio de la siguiente, por ello cuando se produzca alimentaremos los motores en función del movimiento que queremos que hagan.

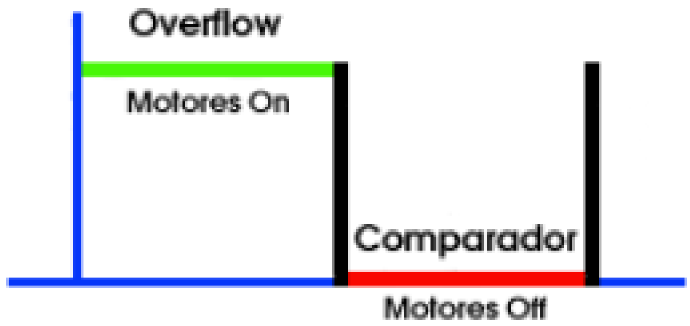

Figura 5.1: Diagrama de alimentación del motor por unidad de tiempo T

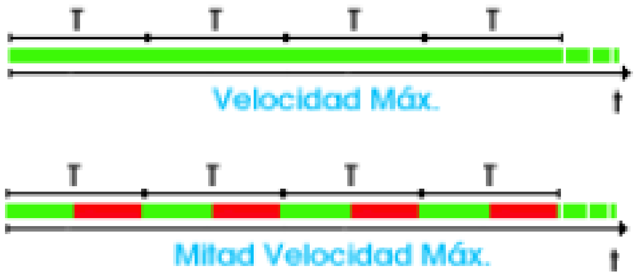

Figura 5.2: Diagrama temporal de alimentación del motor

Para limitar la cantidad de unidad de tiempo que alimentamos los motores, debemos configurar uno de los canales del temporizador como comparador y le introducimos el valor que debe comparar, Z, tal que Z <Y. En el ejemplo se ha configurado el canal 0 como comparador y el valor que se ha introducido para comparar es dinámico con el objetivo de poder comprobar todas las posibles velocidades del motor, todo ello dentro de un bucle, de modo que el motor acelera y desacelera constantemente. Sin embargo, supongamos que Z es 15. Entonces cuando el temporizador llegue a 15 (30/2), o lo que es lo mismo, a los 195/2 useg se producirá la interrupción del comparador el cual nos indica que hemos alimentado los motores ese tiempo, entonces lo que debemos hacer es parar los motores. Esto supondrá que hasta la interrupción de Overflow lo motores estarán parados, y serán alimentados tras producirse esta.

Con este sistema hemos conseguido controlar la tensión que llega a los motores, o lo que es lo mismo, su velocidad.

5.5.3. Ejemplo de control de velocidad

Supongamos la disposición de motores del ejemplo 5.3. Los motores están conectados en J15 y J14. Lo que haremos será un programa que acelerará y decelerará los mismo continuamente.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; CLUB ROBOTICA - MECATRONICA   ETSI - UAM
;;;
;;; SISTEMA DE DESARROLLO GP_Bot
;;;
;;; Programa ejemplo de control de velocidad de motores CC.
;;;
```

```
;;; Queremos controlar la velocidad de dos motores de CC situados en los
;;; conectores J14 y J15.
;;; El programa acelera y decelera los motores continuamente.
;;; Es muy importante conocer los valores que determinan el movimiento de cada
;;; motor, lo cual depende de la conexión de los cables de alimentación.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción
InitStk     EQU $23F      ;Inicio de la pila

;;;;; Incluimos las definiciones de los registros del micro
$Include 'gpregs.inc'

;;; DEFINICION DEL MOVIMIENTO DEL MOTOR
;;; Motor derecho en pines PTA2 y PTA1 (J14)
;;; Motor izquierdo en pines PTA4 y PTA3 (J15)
;;; 01 hacia adelante
;;; 10 hacia atras
ADELANTE    EQU 00001010q ;Ambos motores adelante
ATRAS       EQU 00010100q ;Ambos motores atras
DERECHA     EQU 00001100q ;Derecho atras / izquierdo adelante
IZQUIERDA   EQU 00010010q ;Derecho adelante / izquierdo atras

;;; DEFINICION DE TIEMPOS DE ALIMENTACION
PERIODO      EQU 30T      ;Unidad de tiempo T = 30*6,5 us
LIMITE_A     EQU 28T      ;Valor máximo (velocidad máxima)
LIMITE_D     EQU 2T       ;Valor mínimo (velocidad mínima)

;;; VARIABLES DEL PROGRAMA
             ORG RamStart
operacion     DS 1        ;Indica si aceleramos o deceleramos

;;; PROGRAMA
             ORG RomStart

main:
             ;Inicializamos pila
             LDHX #InitStk
             TXS

             ;Deshabilitamos el COP
             BSET 0,CONFIG1

             ;;; CONFIGURACIÓN DEL TIMER 1
             ;Configurar contador - Registro T1SC
             ;Parar temporizador 1, Reset temporizador 1
             ;Generamos señal de 6,5 us
             ;Bit 7 flag de interrupcion a cualquier valor
             ;Bit 6 habilita la interrupción de Overflow. Bit 6 = 1
             ;Bit 5 para el contador. Bit 5 = 1
```

```
;Bit 4 resetea el contador. Bit 4 = 1
;Bit 3 no usado.
;Bits 2 a 0 configuran el divisor de frecuencia.
;1/(2,45 MHz / 16) = 6,5 useg
;luego Bit 2 = 1 y Bits 1 y 0 = 0
LDA #01110100q
STA T1SC

;Introducimos el periodo del contador en el registro del
;del modulo del temporizador 1. Se trata del valor hasta el
;que debe contar el contador.
LDHX #PERIODO
STHX T1MODH

;Configurar canal 0 del timer 1 como comparador - Registro T1SC0
;Configuramos canal 0 temporizador 1 como comparador
;Bit 7 flag de interrupcion a cualquier valor
;Bit 6 habilita la interrupción. Bit 6 = 1
;Bits 5 y 4 seleccionan el modo.
;Bit 5 = 0 y Bit 4 = 1 seleccionan modo comparador
;Bit 3 y 2 seleccionan flanco/nivel.
;Bit 3 y 2 = 0 seleccionan salida estandard
;Bit 1 invierte salida si Overflow. Bit 1 = 0
;Bit 0 selecciona máximo ciclo de trabajo. Bit 0 = 0
LDA #01010000q
STA T1SC0

;Introducimos el ancho del pulso en el registro del canal 0
;del temporizador 1. Se trata del valor hasta el que alimentamos
;los motores
LDHX #LIMITE_D
STHX T1CH0H

;Activamos temporizador 1
;Bit 5 de stop a 0
LDA T1SC
AND #11011111q
STA T1SC

;;; CONFIGURACION MOTORES
;Configuramos los pines PTA4, PTA3, PTA2 y PTA1 como salida
;El resto de pines no importan (los configuramos como salidas
;también)
LDA #$FF
STA DDRA

;Inicializamos
CLRA
STA PORTA        ;Paramos motores
STA operacion    ;0 es acelerar 1 decelerar

;Habilitamos interrupciones
CLI
```

```
            ;Bucle de aceleracion y deceleracion
   bucle:
            ;Retardo
            ;(depende de los motores del robot)

            ;Comprobamos operacion (si estamos acelerando o decelerando)
            LDA operacion
            CMP #0
            BNE decelerar
            BRA acelerar

            ;Estamos decelerando
   decelerar:
            LDHX T1CH0H
            CPHX #LIMITE_D  ;Comprobamos si hemos decelerado del todo
            BNE decelera    ;Si no es así seguimos decelerando
            CLRA
            STA operacion   ;Si hemos decelerado del todo entonces
            BRA acelerar    ;pasamos a acelerar

            ;Para decelerar decrementamos en dos unidades (por ejemplo)
            ;el valor del tiempo que estamos alimentando
            ;los motores (modulo del comparador)
   decelera:
            AIX #-2T
            STHX T1CH0H
            BRA sigue

            ;Estamos acelerando
   acelerar:
            LDHX T1CH0H
            CPHX #LIMITE_A  ;Comprobamos si hemos acelerado del todo
            BNE acelera     ;Si no es asi seguimos acelerando
            LDA #1
            STA operacion   ;Si hemos acelerado del todo entonces
            BRA decelerar   ;pasamos a decelerar

            ;Para acelerar incrementamos en dos unidades (por ejemplo)
            ;el valor del tiempo que estamos alimentando
            ;los motores (modulo del comparador)
   acelera:
            AIX #2T
            STHX T1CH0H

           ;Volvemos al inicio
   sigue:
            JMP bucle

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; CMP0 - Rutina de Servicio de Interrupcion del comparador
;;; Cuando se produce esta interrupción debemos dejar de alimentar los motores
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   cmp0:
            ; Ponemos a 0 el flag de interrupcion
```

```
            BCLR 7,T1SC0

            ;Guardamos el registro H
            PSHH

            ;Paramos los motores
            CLRA
            STA PORTA

            ;Recuperamos H
            PULH

            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; OVER - Rutina de Servicio de Interrupcion de Overflow del timer 1
;;; Cuando se produce esta interrupción alimentamos los motores en función
;;; del movimiento que queramos hacer.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   over:
            ;Ponemos a 0 el flag de interrupcion
            BCRL 7,T1SC

            ;Guardamos el registro H
            PSHH

            ;Hacemos que los motores vayan para adelante
            LDA #ADELANTE
            STA PORTA

            ;Recuperamos H
            PULH

            RTI


;;; VECTORES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DUMMY-ISR - Rutina de Servicio de Interrupcion "que no hace nada"
;;; Simplemente retornamos
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dummy_isr:
            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Vectores de interrupción.
;;; Indicamos las rutinas de servicio para cada interrupción.
;;; En caso de no emplear alguna, apuntar a la rutina dummy_isr.
;;; El Vector de Reset debe apuntar al inicio del programa.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ORG VectorStart
            DW dummy_isr     ;Vector TMB
            DW dummy_isr     ;Vector DAC
```

```
        DW dummy_isr     ;Vector KBI
        DW dummy_isr     ;Vector Transmisor SCI
        DW dummy_isr     ;Vector Receptor SCI
        DW dummy_isr     ;Vector Error SCI
        DW dummy_isr     ;Vector Transmisor SPI
        DW dummy_isr     ;Vector Receptor SPI
        DW dummy_isr     ;Vector Overflow TIM2
        DW dummy_isr     ;Vector Canal 1 TIM2
        DW dummy_isr     ;Vector Canal 0 TIM2
        DW over          ;Vector Overflow TIM1
        DW dummy_isr     ;Vector Canal 1 TIM1
        DW cmp0          ;Vector Canal 0 TIM1
        DW dummy_isr     ;Vector PLL
        DW dummy_isr     ;Vector IRQ1
        DW dummy_isr     ;Vector SWI
        DW main          ;Vector Reset
```

5.6. Comunicación Serie Asíncrona (SCI)

Cuando queremos comunicar nuestra placa con un PC u otra placa, uno de los medios más sencillos es a través del puerto serie asíncrono (el mismo que tiene el PC). Aquí se presentan las rutinas de escritura y lectura del puerto serie asíncrono. Es posible asociar cada una de las rutinas a una interrupción, aunque la recepción es la única que tiene sentido, ya que podríamos estar realizando cualquier tarea mientras que no lleguen datos y cuando eston llegan, nos avisa la interrupción. Sin embargo, este ejemplo no hace uso de la interrupciones, con lo cual es necesario llamar a la rutina de lectura del puerto serie de forma explícita.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; CLUB ROBOTICA - MECATRONICA   ETSI - UAM
;;;
;;; SISTEMA DE DESARROLLO GP_Bot
;;;
;;; Programa ejemplo de uso del puerto serie asíncrono (SCI).
;;;
;;; Se configura el puerto serie asíncrono para recibir datos de 8 bits
;;; (un byte) a una velocidad de 19200 baudios sin paridad.
;;; Es importante que esta configuración se dé también en el sistema con el
;;; que nos queremos comunicar (PC u otra placa).
;;; El bucle principal del programa lee un byte del puerto serie y lo
;;; devuelve.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; DEFINICION DE LAS POSICIONES DE MEMORIA
RamStart    EQU $0040     ;Inicio de la RAM
RomStart    EQU $8000     ;Inicio de la FLASH
VectorStart EQU $FFDC     ;Inicio de los vectores de interrupción
InitStk     EQU $23F      ;Inicio de la pila

;;;;; Incluimos las definiciones de los registros del micro
$Include 'gpregs.inc'
```

```
;;; VARIABLES DEL PROGRAMA
            ORG RamStart


;;; PROGRAMA
            ORG RomStart

main:
            ;Inicializamos pila
            LDHX #InitStk
            TXS

            ;Deshabilitamos el COP
            BSET 0,CONFIG1

            ;;;CONFIGURACIÓN COMUNICACIÓN SERIE
            ;Configuramos velocidad a 19200 baudios
            ;(ojo que la configuracion se corresponde con
            ;9600 a fbus=4.9152MHz, pero la fbus real es 2.45 Mhz)
            LDA #$21
            STA SCBR

            ;Habilitar el SCI ->1 en bit ENSCI
            ;8 bits ->0 en bit M
            ;Sin paridad ->0 en bits PEN y PTY
            LDA #$40
            STA SCC1

            ;Habilitar el transmisor ->1 en bit TE
            ;Habilitar el receptor ->1 en bit RE
            LDA #$0C
            STA SCC2

            ;Lo que leemos lo devolvemos
            ;Leemos del puerto serie
   lee:
            JSR leer_dato

            ;Escribimos
            JSR escribir_dato

            JMP lee


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Leer_dato - Rutina de lectura del puerto serie asíncrono
;;; El byte leido se devuelve en el acumulador
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;Limpiar el bit SCRF (Receptor lleno) leyendolo
leer_dato:
            BRCLR 5,SCS1,leer_dato

            ;Receptor lleno, leer en el registro SCDR
```

```
            ;el nuevo dato recibido
            LDA SCDR

            RTS

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Escribir_dato - Rutina de escritura en puerto serie asíncrono
;;; Se envia el contenido del acumulador
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;Limpiar el bit SCTE (Transmisor vacio) leyendolo
            ;Si es 1 podemos escribir un nuevo dato
escribir_dato:
            BRCLR 7,SCS1,escribir_dato

            ;Emisor vacio, escribiendo en el registro SCDR
            ;el nuevo dato a enviar
            STA SCDR

            RTS

;;; VECTORES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; DUMMY-ISR - Rutina de Servicio de Interrupcion “que no hace nada”
;;; Simplemente retornamos
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dummy_isr:
            RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Vectores de interrupción.
;;; Indicamos las rutinas de servicio para cada interrupción.
;;; En caso de no emplear alguna, apuntar a la rutina dummy_isr.
;;; El Vector de Reset debe apuntar al inicio del programa.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ORG VectorStart
            DW dummy_isr     ;Vector TMB
            DW dummy_isr     ;Vector DAC
            DW dummy_isr     ;Vector KBI
            DW dummy_isr     ;Vector Transmisor SCI
            DW dummy_isr     ;Vector Receptor SCI
            DW dummy_isr     ;Vector Error SCI
            DW dummy_isr     ;Vector Transmisor SPI
            DW dummy_isr     ;Vector Receptor SPI
            DW dummy_isr     ;Vector Overflow TIM2
            DW dummy_isr     ;Vector Canal 1 TIM2
            DW dummy_isr     ;Vector Canal 0 TIM2
            DW dummy_isr     ;Vector Overflow TIM1
            DW dummy_isr     ;Vector Canal 1 TIM1
            DW dummy_isr     ;Vector Canal 0 TIM1
            DW dummy_isr     ;Vector PLL
            DW dummy_isr     ;Vector IRQ1
            DW dummy_isr     ;Vector SWI
            DW main          ;Vector Reset
```

Capítulo 6

Apéndices

6.1. Apéndice A: El fichero 'gpregs.inc'

Este fichero incluye la definición de todos los registros de configuración, control, etc. del microcontrolador. Se corresponde con el fichero original que se puede encontrar en el directorio principal del entorno de programación.

```
 ; 68HC908GP20/GP32 Equates

PTA     EQU $0000     ; Ports and data direction
PORTA   EQU $0000
PTB     EQU $0001
PORTB   EQU $0001
PTC     EQU $0002
PORTC   EQU $0002
PTD     EQU $0003
PORTD   EQU $0003
DDRA    EQU $0004
DDRB    EQU $0005
DDRC    EQU $0006
DDRD    EQU $0007
PTE     EQU $0008
PORTE   EQU $0008
DDRE    EQU $000C

PTAPUE  EQU $000D     ; Port pull-up enables
PTCPUE  EQU $000E
PTDPUE  EQU $000F

SPCR    EQU $0010     ; SPI (Syncronous communications)
SPSCR   EQU $0011
SPDR    EQU $0012

SCC1    EQU $0013     ; SPI (Asyncronous communications)
SCC2    EQU $0014
SCC3    EQU $0015
SCS1    EQU $0016
SCS2    EQU $0017
SCDR    EQU $0018
```

```
SCBR    EQU $0019

INTKBSCR EQU $001a   ; Keyboard interrupt control/status
INTKBIER EQU $001b

TBCR    EQU $001c    ; Time base module

INTSCR  EQU $001d    ; IRQ status/control

CONFIG2 EQU $001e    ; System configuration
CONFIG1 EQU $001f

T1SC    EQU $0020    ; Timer 1
T1CNTH  EQU $0021
T1CNTL  EQU $0022
T1MODH  EQU $0023
T1MODL  EQU $0024
T1SC0   EQU $0025
T1CH0H  EQU $0026
T1CH0L  EQU $0027
T1SC1   EQU $0028
T1CH1H  EQU $0029
T1CH1L  EQU $002a

T2SC    EQU $002b    ; Timer 2
T2CNTH  EQU $002c
T2CNTL  EQU $002d
T2MODH  EQU $002e
T2MODL  EQU $002f
T2SC0   EQU $0030
T2CH0H  EQU $0031
T2CH0L  EQU $0032
T2SC1   EQU $0033
T2CH1H  EQU $0034
T2CH1L  EQU $0035

PCTL    EQU $0036    ; Phase lock loop (for crystals)
PBWC    EQU $0037
PMSH    EQU $0038
PMSL    EQU $0039
PMRS    EQU $003A
PMDS    EQU $003B

ADSCR   EQU $003C    ; A to D converter
ADR     EQU $003D
ADCLK   EQU $003E

SBSR    EQU $fe00    ; System integration
SRSR    EQU $fe01
SUBAR   EQU $fe02
SBFCR   EQU $fe03

INT1    EQU $fe04    ; Interrupt status
INT2    EQU $fe05
```

```
INT3    EQU $fe06

FLTCR   EQU $fe07    ; Flash test/programming
FLCR    EQU $fe08

BRKH    EQU $fe09    ; Hardware breakpoint
BRKL    EQU $fe0a
BRKSCR  EQU $fe0b

LVISR   EQU $fe0c    ; Low voltage detect

FLBPR   EQU $ff80    ; Flash boot protect

COPCTL  EQU $ffff    ; COP (Computer operating properly) control


;(C)opywrite P&E Microcomputer Systems, 1998
; You may use this code freely as long as this copyright notice
; is included.
```