Escuela Superior de Ingenieros
Universidad de Sevilla

PROYECTO FIN DE CARRERA

"Diseño de un Sistema de Control para un Motor de Continua basado en DSP"

Volumen II

Realizado Por:
Juan Carlos del Pino López
Tutor del Proyecto:
Pedro L. Cruz Romero

Índice general de contenidos

VOLUMEN II

Índice de Figuras y Tablas

VOLUMEN II

CAPÍTULO 4: REGISTROS Y PERIFÉRICOS DEL TMS320F243

CAPÍTULO 5: APLICACIÓN DE LOS DSP´S AL CONTROL DE MOTORES

CAPÍTULO 4: REGISTROS Y PERIFÉRICOS DEL TMS320F243

4.1. INTRODUCCIÓN:

Una de las principales ventajas de los DSP frente a los microcontroladores habituales era la inclusión en el propio chip de una serie de periféricos muy empleados en aplicaciones de control digital, lo cual abarataba los costes al reducir el número de componentes necesarios para montar un sistema de control, además de optimizar el rendimiento al aumentar la velocidad de comunicación y reducir las posibilidades de ruidos e interferencias en las mismas. Por ello, se hace necesaria la inclusión de un capítulo expresamente dedicado a la descripción del funcionamiento y programación de cada uno de los periféricos y registros que controlan la forma de trabajar del DSP. En lo que sigue, serán descritos los principales registros y periféricos que han sido estudiados en éste proyecto, aunque no se comentarán todas las posibilidades que proporcionan algunos, pues quedan fuera de los objetivo de este documento.

Para el correcto funcionamiento del DSP, es necesaria la programación de una serie de registros, ya sean de control o de algunos periféricos. Éstos suelen estar divididos a su vez en distintos campos en los que se configuran algunas de las prestaciones o posibilidades del DSP. Por ello, a lo largo de éste capítulo, se hará referencia a distintas formas de programar y configurar éstos registros empleado distintos lenguajes de programación, de forma que el lector vaya familiarizándose con el modo de trabajo con los DSP´s. Todos los registro del DSP susceptibles de ser programados se encuentran listados en el Anexo 2, en el que se indica la dirección y una descripción de cada registro. También se incluye en dicho anexo el archivo "*regs243.h*", el cual será empleado en los ejemplos que se describen en éste capítulo y en el que se definen las etiquetas que se emplearán para hacer referencia a cada registro sin tener que conocer las direcciones de cada uno para emplearlos. Ello se consigue mediante la directiva "#define" como se indicó en capítulos anteriores, como por ejemplo:

```
#define IMR *(volatile unsigned int *)0x0004 /*Interrupt Mask
                                                Register */
```

Se emplea el lenguaje C debido a que los ejemplos que se pondrán en práctica están escritos en dicho lenguaje por su facilidad frente al lenguaje ensamblador.

Para poner en práctica los conocimientos detallados sobre cada periférico, se detallarán una serie de programas de ejemplos que ayuden a entender el funcionamiento de cada uno. Ello se hará a través de dos programas: Code Composer, para programoación en lenguajes de alto nivel, y VisSim - Embedded Controls Developer v5.0 for TI C2000, que facilita la tarea al realizarse mediante conexión de bloques ya definidos. Para llevar a cabo éstos ejemplos, se ha desarrollado una placa de pruebas en la que se van a comprobar el funcionamiento de cada uno de ellos (figura 4.1). Dicha placa ha sido desarrollada en base a los ejemplos y tutoriales proporcionados por Texas Instruments, así como ejemplos desarrollados por otras universidades. En ella se engloban varios de ellos de forma que se disponga de una única plataforma de pruebas en la que testear el funcionamiento de todos los periféricos del DSP. Sus características principales se detallan a continuación.

4.2. PLACA DE PRUEBAS:

La placa de pruebas desarrollada (figura 4.1), pretende ser una interfaz en la que se tengan accesibles las distintas señales de los periféricos del DSP de forma sencilla. En ella se busca tener separadas dichas señales según los periféricos o grupos de señales semejantes. Su conexión al DSP se hace a través de los conectores de expansión P1 (interfaz I/O) y P2 (Interfaz analógica) del mismo descritos anteriormente, mediante cables de banda y conectores de 40 y 20 pines respectivamente (figura 4.2). De esta forma se llevan las señales más importante del DSP a la placa, donde se distribuyen en grupos de pines según su funcionalidad: Puertos de señales digitales de Entrada/Salida y entradas a los 8 canales del convertidor A/D. Así, si es necesario, cualquiera de ellas puede conectarse a la sonda de un osciloscopio para ser monitorizadas

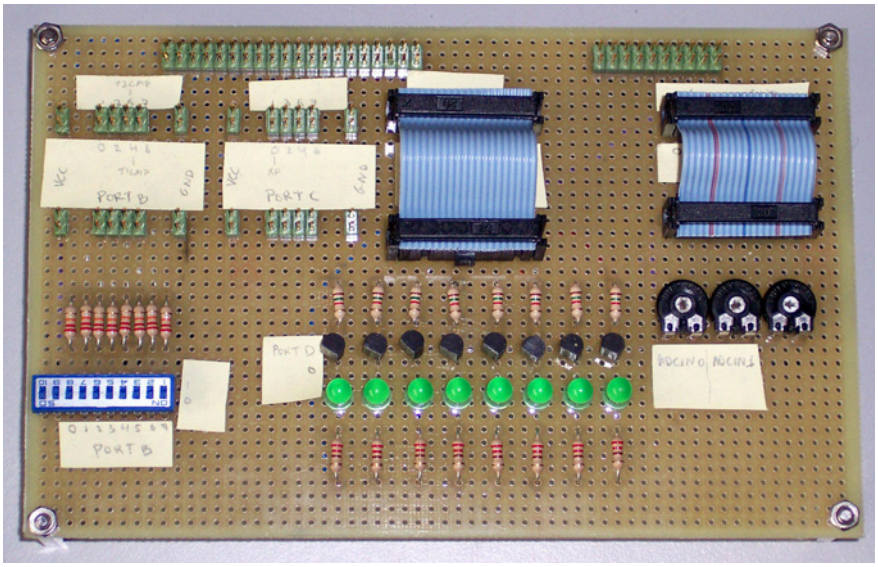


Figura 4.1. Placa de pruebas.

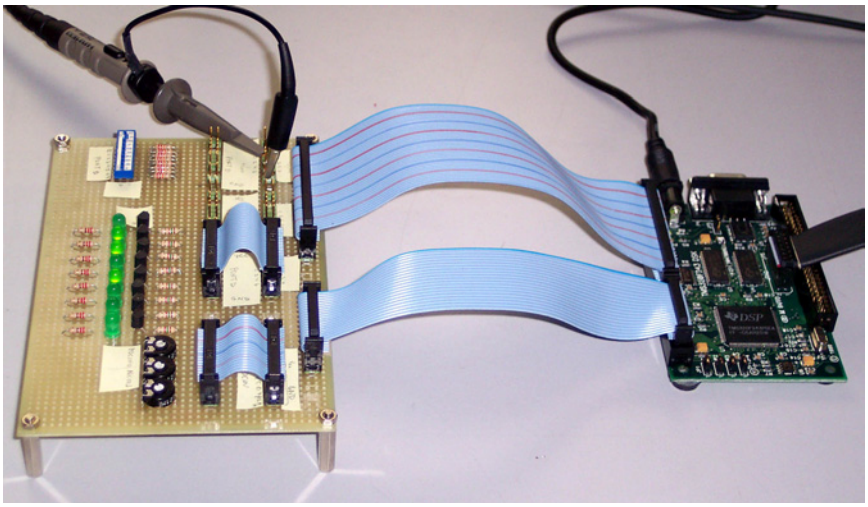


Figura 4.2. Conexión al DSP.

Como se aprecia en la figura 4.1, la división de pines se ha hecho conforme al orden que tienen estas señales en los conectores de expansión P1 Y P2, como se vio con anterioridad (tablas 2.9 y 2.10). Por un lado, se obtienen directamente un grupo de 20 pines que corresponden al conector de expansión analógico. Por otro, del conector de Entrada/Salida P2, se sacan 3 grupos de 12 pines correspondientes a las señales de Entrada/Salida de los puertos digitales B, C y D, ya que en el conector P2 se encuentran ordenadas las señales según su función principal después de un reset del DSP, que en su mayoría corresponden a dichas señales E/S. El resto de señales, junto con las del puerto A, no se han tomado al no ser empleadas en los ejemplos que se emplean en este capítulo. En los cuatro grupos de pines, además de las señales comentadas anteriormente, están disponibles pines en los que se encuentran la alimentación a +5V (V_{CC}) y la señal de tierra (GND). Se ha tomado esta estructura para facilitar la conexión de elementos de prueba que requieran de estas señales, como son leds, potenciómetros, etc. La distribución de señales en cada grupo es la siguiente:

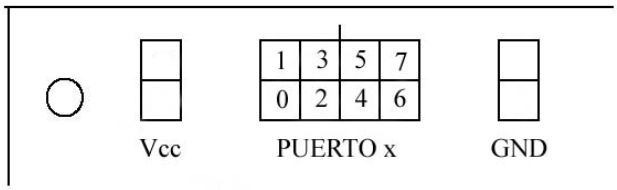


Figura 4.3. Distribución de pines E/S en la placa.

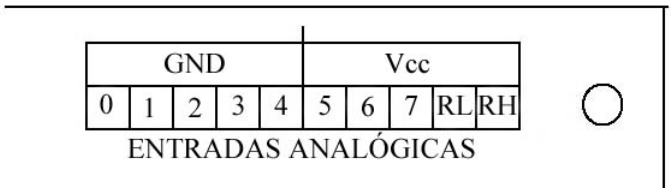


Figura 4.4. Distribución de entradas analógicas.

Donde RL es la tensión baja de referencia del convertidor analógico-digital y RH la referencia de tensión alta.

Para la comprobación de distintos experimentos, la placa de pruebas ha sido dotada de una serie de elementos para visualizar y comprobar de manera sencilla la funcionalidad de los mismos. Esto se ha conseguido mediante la conexión de cables de banda de 20 pines, que hacen las veces de puente de unión, y que permiten la conexión o no de dichos componentes de prueba a los 4 grupos de señales descritos anteriormente (figura 4.1). Los elementos que se han incluido para la realización de experimentos son:

- **Puerto B de E/S digitales:**
  Se dispone de un grupo de 10 interruptores en formato DIP que, mediante unas resistencias pullup de 22k, conectan las señales de dicho puerto a +5V o a 0V, lo que equivale a poner dichas señales a valores lógicos de "1" o "0". Éstos interruptores se emplean únicamente cuando las señales del puerto B se están utilizando como **entradas digitales**. Dicho puerto sólo tiene 8 señales, por lo que existen 2 interruptores que no se emplean.

- **Puerto C de E/S digitales:**
  En este grupo de señales no se he dispuesto ningún tipo de elemento, de forma que pueda ampliarse en un futuro con lo que se crea necesario.

- **Puerto D de E/S digitales:**
  En este puerto se ha dispuesto un banco de leds, los cuales son controlados por transistores npn de propósito general. Dichos transistores son controlados por el puerto a modo de interruptores que encienden o apagan dichos leds. Para ello, cada señal del puerto se conecta a la base de un transistor mediante una resistencia de 1,5k, y el emisor de cada uno se conecta a tierra. Por último, cada colector se conecta al cátodo de los leds y el ánodo a la tensión de alimentación mediante resistencias de 220 Ohmios. Este montaje se emplea únicamente cuando las señales de éste puerto van a emplearse como **salidas digitales**.

- **Entradas analógicas:**
  Aquí se tiene acceso a los 8 canales de entrada del convertidor analógico-digital. Para la experimentación se han conectado 3 potenciómetros en los canales 0, 3 y 5. De esta forma se puede simular la entrada de tensiones en el convertidor que varían desde 0V a +5V.

Es recomendable que cada uno de estos grupos se conecte con el cable de banda después de ejecutar y parar una vez la aplicación en el DSP, de forma que las distintas señales se configuren como entradas o salidas. De esta forma se evita conectar accidentalmente a tensión algún pin que vaya a funcionar como salida, lo que podría dañar el DSP. De igual forma, es recomendable desconectar los grupos de elementos que no vayan a ser empleados en la aplicación que se va a ejecutar.

El esquema seguido para el montaje de los distintos elementos es el siguiente (figura):

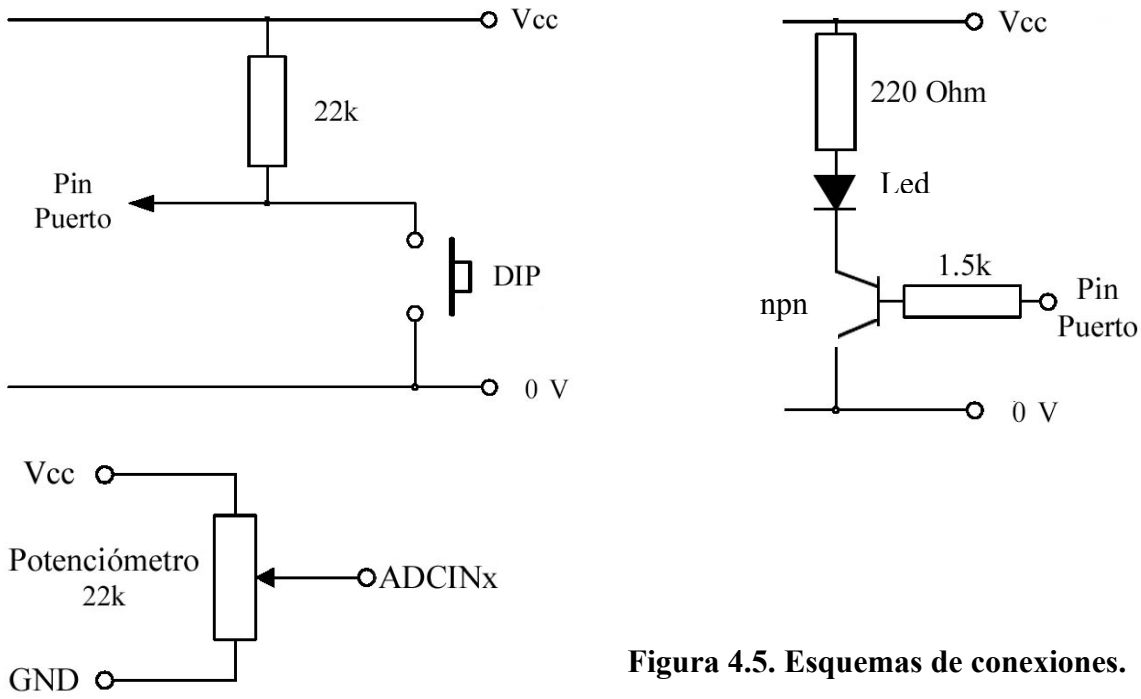


Figura 4.5. Esquemas de conexiones.

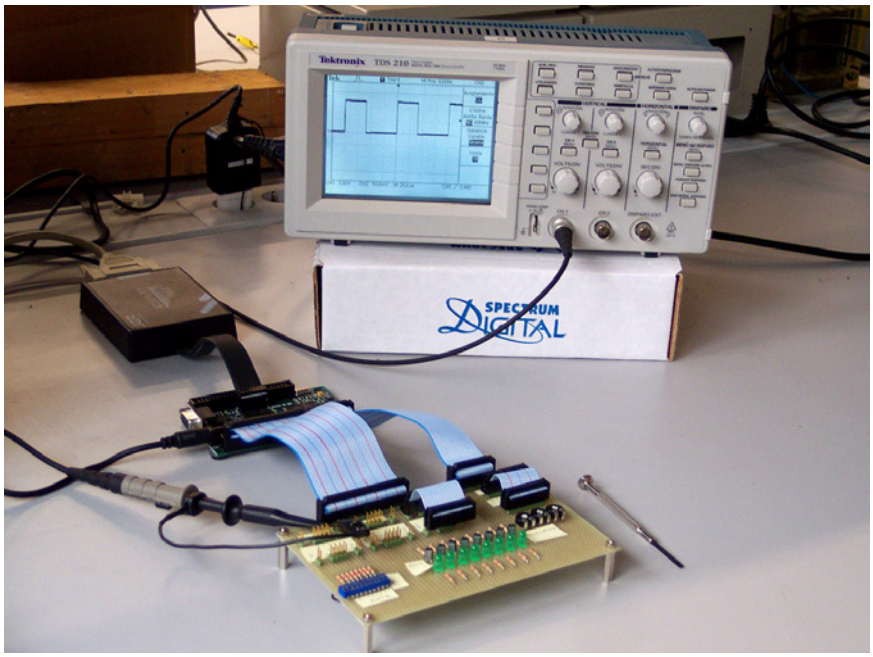


Figura 4.6. Generación de ondas PWM.

A través de éstos grupos de dispositivos, se pueden poner práctica la operación con algunos de los periféricos más destacables del DSP, como son:

- Lectura y escritura a través de las señales de Entrada/Salida digitales.
- Implementación de algoritmos para controlar el valor de las salidas digitales.
- Generación de ondas cuadradas o PWM.
- Lectura de varias señales a través del convertidor analógico-digital.
- Control de las salidas digitales mediante medidas analógicas.
- Control de señales PWM a partir de medidas analógicas.

Estas posibilidades serán puestas en práctica a través de los distintos ejemplos que se desarrollarán a lo largo de cada periférico (figura 4.6). Ello se llevará a cabo mediante los dos programas de desarrollo detallados en el capítulo anterior: Code Composer y VisSim - Embedded Controls Developer v5.0 for TI C2000. De esta forma se verán las diferencias a la hora de realizar aplicaciones con los dos programas, además de clarificar el uso y posibilidades de cada periférico. Por ello, es recomendable tener claro los pasos principales para poner en práctica una aplicación con ambos programas, como se detallaron en el capítulo anterior:

- Abrir o generar un proyecto en Code Composer.
- Compilar y cargar el proyecto con Code Composer.
- Programar la aplicación en el DSP con Code Composer.
- Abrir o realizar un diagrama en Vissim.
- Configurar las opciones adecuadas para la simulación en vissim.
- Simular el diagrama en modo interactivo o compilado en Vissim.
- Representación gráfica en Vissim.

4.3. PRINCIPALES REGISTROS DE INICIACIÓN:

A la hora de poner en práctica una aplicación, es necesario inicializar y configurar el funcionamiento del DSP. Para ello se deben configurar una serie de registros, los cuales pueden influir en gran medida en lo que a la ejecución de la aplicación se refiere. Éste paso suele ser el primero en llevarse a cabo en toda programación de una aplicación, y consiste en:

- Iniciar los registros de control del sistema y DSP.
- Configurar la generación de tiempos de espera.
- Configurar el temporizador de guarda Watchdog.
- Configurar e iniciar los registros de control de cada periférico.

Los tres primeros pasos se detallan a continuación, mientras el último será descrito a lo largo de la exposición de los distintos periféricos. Resaltar que, como se vio en el capítulo 2 donde se analizaba el mapa de memoria del TMS320F243, la mayor parte de los registros de configuración del DSP y de sus periféricos se encontraban mapeados en la memoria de datos.

Destacar, que la configuración e iniciación de estos registro las tiene que llevar a cabo el programador cuando desarrolle una aplicación mediante Code Composer. Si la aplicación se desarrolla con Vissim no es necesario hacerlo, ya que el mismo programa se encarga de configurar el sistema como crea conveniente según el uso de cada periférico en el diagrama.

4.3.1. Registro de control y estado del sistema (SCSR):

Éste registro de control del DSP debe ser de los primeros en configurarse, ya que se encarga principalmente de establecer el modo de bajo consumo del DSP. Para ello, el procesador TMS320F243, dispone de una instrucción “IDLE” que se encarga de parar todos los relojes internos de la CPU, mientras que la salida de reloj externa (CLKOUT) continúa funcionando, además de no cambiar el estado de los pines de Entrada/Salida (GPIO). De esta forma se consigue entrar en un modo de bajo consumo, reduciendo considerablemente la potencia consumida por el DSP. De éste estado puede salirse únicamente mediante un reset o por una petición de interrupción. Entrar en un modo de bajo consumo se consigue actuando sobre el funcionamiento de los dos dominios de actuación que tiene el reloj en las unidades internas del procesador:

- **Dominio del reloj de la CPU:** Consiste en los relojes que actúan en el circuito lógico de la unidad CPU.
- **Dominio del reloj del sistema:** Es el reloj de los periférico, como las derivadas de la señal CLKOUT de la CPU y el reloj que maneja la lógica de interrupción en la CPU.

Existen tres modos de bajo consumo, en cada uno de los cuales el consumo del procesador es menor:

- ***IDLE1 (LPM0):*** se para el reloj de la CPU mientras el reloj del dominio del sistema continúa funcionando.

- ***IDLE2 (LPM1):*** tanto el reloj de la CPU como el del dominio del sistema se detienen.
- ***HALT (LPM2):*** se detienen el reloj de la CPU y el del dominio del sistema además del oscilador del temporizador de guardia Watchdog.

La forma de entrar en estos modos de bajo consumo se realiza mediante los bits de control LPM1 Y LPM0 del registro SCSR. El significado de todos los bits de éste registro, mapeado en la dirección de memoria de datos 7018h, son:

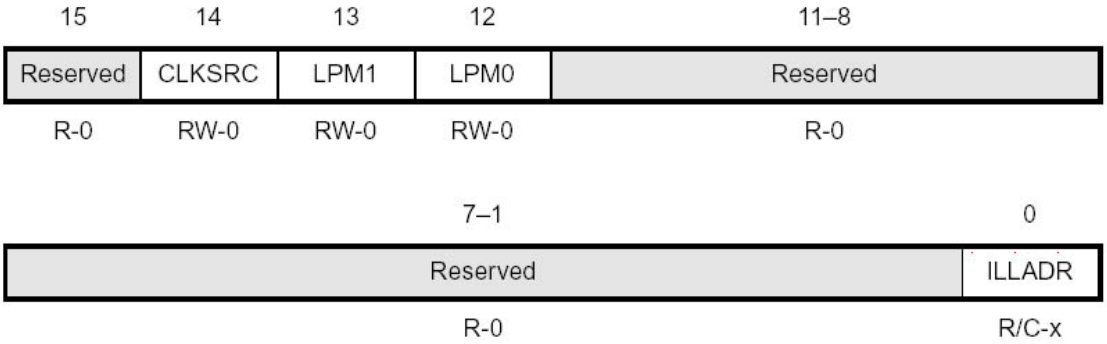


Figura 4.7. Registro de control SCSR (7018h).

Nota: R: acceso de lectura; W: acceso de escritura; C: solo se puede poner a '0'; Después de "-" sigue el valor después de un reset.

- ***Bit 15: Reservado.*** Si se lee siempre devuelve como valor cero.

- ***Bit 14: CLKSRC.*** Selección de la señal de reloj en el pin de salida de reloj:

  0 En el pin de salida está el reloj de la CPU CLKOUT (20Mhz).
  1 El pin de salida es el reloj del temporizador de guardia Watchdog WDCLK (39062.5 Hz).

- ***Bits 13-12: LPM(1:0).*** En estos bits se indica el modo de bajo consumo seleccionado para cuando se ejecute un instrucción IDLE. Sus valores son:

Bits LPM(1:0)	Modo de bajo consumo
xx	CPU normal
00	IDLE1 (LPM0)
01	IDLE2 (LPM1)
1x	HALT (LPM2)

- ***Bits 11-1: Reservado.*** Se leen a cero siempre.

- ***Bit0: ILLADR.*** Bit de detección de dirección ilegal. Se pone a '1' cuando se intenta acceder a una dirección ilegal o errónea. Este bit se borra escribiendo un '1' en él. No se limpia este bit en un reset. Una dirección ilegal puede causar una interrupción no enmascarable NMI.

4.3.2. Temporizador de guarda Watchdog (WD):

Uno de los periféricos más empleados actualmente en cualquier tipo de procesadores es el denominado temporizador de guarda o Watchdog. Éste dispositivo también se incluye en el TMS320F243, y se encarga de monitorizar tanto el software como el hardware del sistema. Su función consiste en detectar posibles fallos debido a perturbaciones eléctricas (EMI), fallos en el programa que se está ejecutando, etc. y si es el caso, resetear el sistema. Para ello dispone de un registro temporizador (WDCNTR), el cual funciona de forma independiente a la CPU, en el que se escribe el valor adecuado del tiempo que debe transcurrir antes de ejecutar un reset. El watchdog se activa inmediatamente después de un reset, a partir del cual el temporizador va incrementando su valor una unidad en cada ciclo del reloj del Watchdog (WDCLK) hasta llegar al desbordamiento, momento en el que se realiza el reset del sistema. Éste reloj proviene de la señal de reloj del procesador CPUCLK (de 20Mhz para el TMS320F243), y es de menor frecuencia que éste, según la expresión:

$$WDCLK = (CLKOUT)/512$$

Por lo que su valor es de 39062.5 Hz. La señal de éste reloj puede verse a través del pin CLKOUT si el watchdog está activado.

Las características más notables del temporizador watchdog son:

- Contador de 8 bits cuyo rebasamiento produce el reset del sistema.
- Prescaler con 6 posibles selecciones del reloj del watchdog.
- Registro de reseteo del watchdog (WDKEY), en el que escribiendo una clave correcta se pone a cero el contador WDCNTR. Genera un reset del sistema si la clave es incorrecta.
- Bit de bandera que indica si se ha realizado un reset debido al watchdog.
- Activación automática cuando se realiza un reset hardware en el sistema..

Para la programación del watchdog se disponen de tres registros para el manejo de sus operaciones y control. Éstos son:

- ***WDCNTR:*** Éste registro contiene el valor del contador del temporizador watchdog.
- ***WDKEY:*** Éste registro pone a cero el registro contador WDCNTR cuando se escribe el valor 55h seguido del valor AAh en el registro WDKEY.
- ***WDCR:*** Éste registro contiene los bits para el control del temporizador Watchdog, usados para su configuración inicial, como son:
  - Bit de deshabilitación.
  - Bit del flag del watchdog.
  - Bits de chequeo del watchdog.
  - Bits de selección del prescaler.

4.3.2.1. Registro WDCNTR:

Es un registro de 8 bits que contiene el valor actual del contador del watchdog. Este registro se va incrementando continuamente por cada pulso del reloj interno (WDCLK) del temporizador del watchdog. Únicamente puede ponerse a cero mediante la escritura correcta de la secuencia 55h y AAh en el registro WDKEY. Si este contador llega al rebosamiento (overflow) se realiza un reset del sistema.

Cada uno de los bits de éste registro, mapeado en la dirección de memoria de datos 7023h, tienen el siguiente significado:

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 4.8. Registro contador del WD (7023h).

Nota: R: acceso de lectura; Después de "-" sigue el valor después de un reset.

- ***Bits 7-0: D7 – D0.*** Bits que contienen el valor actual del contador del watchdog. No tiene efecto la escritura en estos bits. Después de un reset valen cero.

4.3.2.2. Registro WDKEY:

El registro WDKEY (WD Reset Key Register) provoca el borrado o puesta a cero del contador del temporizador del watchdog (WDCNTR) cuando se escribe en él el valor 55h seguido de AAh. Cualquier otro valor realiza un reset del sistema. El significado de sus bits es:

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 4.9. Registro WDKEY (7025h).

Nota: R: acceso de lectura; W: acceso de escritura; Después de "-" sigue el valor después de un reset.

- ***Bits 7-0: D7 – D0.*** Éstos bits son solo de escritura y contienen el valor de la secuencia que hay que escribir para producir el borrado del contador del watchdog. Al leerlos no devuelven el último dato escrito. Valen cero depués de un reset.

4.3.2.3. Registro WDCR:

Éste registro se encuentra mapeado en la dirección de memoria de datos 7029h, y contiene los bits de control para la configuración del watchdog. En él se incluyen bits de flag que indican si el watchdog ha reseteado el sistema; bits de chequeo que producen un reset del sistema si se escribe un valor incorrecto en el registro WDCR; y bits para seleccionar el prescaler del reloj de entrada al watchdog. Destacar que para borrar cualquiera de los bits de éste registro debe escribirse un '1' en el bit especificado.

El significado de cada uno de los bits del registro WDCR se muestran en la siguiente figura:

7	6	5	4	3	2	1	0
Reserved	WDDIS	WDCHK2	WDCHK1	WDCHK0	WDPS2	WDPS1	WDPS0
RC-x	RWc-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 4.10. Registro WDCR (7029h).

Nota: R: acceso de lectura; W: acceso de escritura; Wc: acceso de escritura condicionado a que el pin VCCP o WDDIS estén a '1'; C: se pone a '0' al escribir un '1'; Después de "-" sigue el valor después de un reset.

- ***Bit 7: Reservado.***

- ***Bit 6: WDDIS***. Éste bit deshabilita el watchdog siempre que el pin VCCP (en procesadores con memoria FLASH) o el pin WDDIS (en dispositivos con memoria ROM) estén a nivel alto. En el Starter Kit DSK F243 ésta condición equivale a situar el jumper JP1 de la tarjeta en la posición 1-2 (Watchdog disable, como se comentó en el capítulo 2).

  0 Watchdog habilitado.
  1 Watchdog deshabilitado.

- ***Bit 5: WDCHK2.*** En éste bit se debe escribir un '1' cuando se escribe en el registro WDCR, sino se produce una petición de reset del sistema. Siempre es leído como '0'.

  0 Se produce un reset del sistema.
  1 Continúa en operación normal si los demás bits de chequeo se escriben correctamente.

- ***Bit 4: WDCHK1.*** En éste bit se debe escribir un '0' cuando se escribe en el registro WDCR, sino se produce una petición de reset del sistema. Siempre es leído como '0'.

  0 Continúa en operación normal si los demás bits de chequeo se escriben correctamente.
  1 Se produce un reset del sistema.

- ***Bit 3: WDCHK0.*** Éste bit se debe escribir un '1' cuando se escribe en el registro WDCR, sino se produce una petición de reset del sistema. Siempre es leído como '0'.

  0 Se produce un reset del sistema.
  1 Continúa en operación normal si los demás bits de chequeo se escriben correctamente.

- ***Bits 2-0: WDPS2-WDPS0.*** Estos bits seleccionan el prescaler del reloj de entrada al watchdog. De esta forma el reloj interno del watchdog pasa a valer WDCLK/divisor. Recordar que para producirse un rebosamiento del temporizador deben realizarse 257 cuentas dependientes del reloj de entrada al watchdog. Estos bits son de lectura y escritura y cuando se produce un reset se ponen a '0'. En la siguiente tabla se indican las posibles selecciones y los tiempos mínimos para darse un reset en cada caso:

Bits de selección de prescaler del WD				**WDCLK = 39.0625 kHz**	
WDPS2	**WDPS1**	**WDPS0**	**Divisor WDCLK**	**Frecuencia desbordamiento (Hz)**	**Tiempo desbordamiento mínimo (ms)**
0	0	X	1	152.59	6.55
0	1	0	2	76.29	13.11
0	1	1	4	38.15	26.21
1	0	0	8	19.07	52.43
1	0	1	16	9.54	104.86
1	1	0	32	4.77	209.72
1	1	1	64	2.38	419.43

Recordar que para poder escribir en el registro WDCR, siempre debe escribirse la clave '101' en los bits de chequeo WDPSx. Lo contrario produciría un reset del sistema.

Destacar que, en el Starter Kit, el situar el jumper JP1 en la posición 1-2 no deshabilita directamente el watchdog como parece indicar la documentación de la tarjeta, sino que posibilita que sea deshabilitado vía software configurando el bit WDDIS. Por ello, aunque el jumper esté en dicha posición, si no se deshabilita mediante este bit, el watchdog está totalmente activo, ya que lo hace de forma automática después de un reset. Esto es de gran importancia debido a que, si no se configura correctamente el watchdog, la aplicación no se ejecutará correctamente al ser reseteado el DSP por el watchdog con periodicidad. Éste es un error muy común que no suele detectarse, por lo que hace perder el tiempo al programador debido a que suele confundirlo con un error en la aplicación programada

A la hora de programar una aplicación con el watchdog activado, debe realizarse algún tipo de función macro que se dedique a resetear el contador del WD cada cierto tiempo, de forma que este nunca llegue al rebosamiento y ejecute un reset del sistema. Ejemplo de cómo se puede realizar esto se encuentra en el archivo "Boot.asm".

4.3.3. Generador de estados de espera:

Cuando el procesador se comunica con memoria o periféricos externos que son más lentos que él es necesaria la generación de estados de espera. Éste dispositivo añade estados de espera que hacen que la CPU espere ciclos extra de reloj durante un tiempo para que al periférico externo le dé tiempo a acceder a los datos u otros eventos. Se pueden configurar hasta 7 estados de espera en cada uno de los distintos espacios en que se divide la memoria. Ello se consigue configurando al principio del programa el registro de control del generador de estados de espera o **WSGR**, que se encuentra en la dirección **FFFFh del espacio de memoria I/O**:

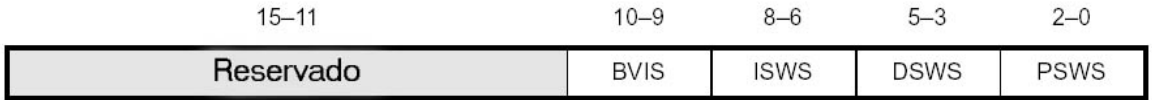


Figura 4.11. Registro WSGR (FFFFh).

- ***Bits 15-11: Reservado.*** Si se leen están siempre a cero.

- ***Bits 10-9: BVIS (Bus Visibility Modes).*** Estos bits permiten la selección de varios modos de visibilidad del bus mientras está en ejecución el programa desde memoria interna, sean datos o programas. Los modos de visibilidad son:

Bit 10	Bit 9	Modo Visibilidad
0	0	Visibilidad del BUS a OFF (reduce ruido y potencia).
0	1	Visibilidad del BUS a OFF (reduce ruido y potencia).
1	0	Visualizar externamente datos y direcciones internas del bus de datos.
1	1	Visualizar externamente datos y direcciones internas del bus de programas.

- ***Bits 8-6: ISWS (I/O space wait-states bits).*** Bits para configurar de 0 (000) a 7 (111) estados de espera en el espacio externo de memoria I/O. Después de un reset contiene el valor '111' que equivale 7 estados de espera.

- ***Bits 5-3: DSWS (Data space wait-states bits).*** Bits para configurar de 0 (000) a 7 (111) estados de espera en el espacio externo de memoria de datos. Después de un reset contiene el valor '111' que equivale 7 estados de espera.

- ***Bits 2-0: PSWS (Program space wait-states bits).*** Bits para configurar de 0 (000) a 7 (111) estados de espera en el espacio externo de memoria de programas. Después de un reset contiene el valor '111' que equivale 7 estados de espera.

4.3.4. Ejemplo de programación de registros:

La configuración de registros de control de cualquiera de los periféricos durante la programación de la aplicación se puede realizar de varios modos:

- Grabando en él directamente el valor que queremos asignarle en hexadecimal.
- Definiendo uno a uno los bits de cada registro y grabándolos en él mediante una suma de números binarios.

Éstos métodos son los que serán aplicados a lo largo de los distintos ejemplos que se introducirán en el análisis de cada periférico. Por ello, todo lo que aquí se detalla, será aplicado en los siguientes apartados. Para facilitar la comprensión, las explicaciones se basarán principalmente en el empleo del lenguaje de programación C en la configuración de registros, aunque se hará mención a determinadas rutinas que puedan realizarse en lenguaje ensamblador por diversas causas.

Para explicar ambos métodos se van a utilizar dos de los registros recién analizados: *SCSR* y *WDCR*. Lo primero que ha de hacerse, como ya se comentó en el capítulo 3, es definir nombres o etiquetas que serán empleados como sustitución de los valores de la dirección de cada registro. De esta forma se emplearán únicamente nombres en lugar de direcciones para la programación, lo cual es mucho más sencillo a la hora de programar que conocer las direcciones de los registros. Estas definiciones se podían hacer en archivos separados (denominados de cabecera ".h") o al principio del mismo archivo principal, y que en lenguaje C tenían la siguiente estructura:

```
#define SCSR    *(volatile unsigned int *)0x7018 /* System
                                                 Control Register */

#define WDCR    *(volatile unsigned int *)0x7029  /* Watchdog
                                                 Control Register */
```

Así, siempre que se quiera hacer referencia o acceder a la posición de memoria 0x7029 del registro de control del watchdog se empleará la etiqueta *WDCR*. Ocurre lo mismo con la dirección 0x7018 y el registro *SCSR*. Una vez definidas las etiquetas, la escritura de un cierto valor en un registro es muy sencilla. Basta con asignarle ese valor a la etiqueta, como por ejemplo:

```
SCSR = 0x0001;
/*
 bit 15        0:      reservado
 bit 14        0:      CLKOUT = CPUCLK
 bit 13-12     00:     modo IDLE1 de bajo consumo
 bit 11-1      0:      reservado
 bit 0         1:      Borra el bit ILLADR
*/

WDCR = 0x00E8;
/*
 bits 15-8     0's:     reservado
 bit 7         1:       borra el flag del WD
 bit 6         1:       deshabilita el WD
 bit 5-3       101:     debe escribirse como 101
 bit 2-0       000:     WDCLK divider = 1
*/
```

De esta manera quedan configurados ambos registros de control con los valores indicados mediante el primero de los métodos mencionados anteriormente. Para su comprensión por parte de otras personas ajenas al programador, éste método requiere de la adición de líneas de comentario indicando el significado de cada bit.

Existe un segundo método que, aunque es más extenso, es más sencillo a la hora de hacer modificaciones en una parte del registro. También permite comprender mejor la configuración que se ha decidido dar a dicho registro, ya que permite definir uno a uno los distintos bits o secciones de registro indicando su significado. De esta manera, basta modificar el valor de uno o varios bits para cambiar la configuración de un registro, y todo ello sin tener que buscar el nuevo valor en hexadecimal. Para ello, además de definir etiquetas para las direcciones, se pueden definir etiquetas para cada uno de los bits o secciones del registro en un archivo ".h" o al principio del programa principal, como por ejemplo:

```
/**************** SETUP for the WDCR-Register  ****************/
#define WDDIS         1     /* 0 : Watchdog enabled 1: disabled*/
#define WDCHK2        1     /* 0 : System reset   1: Normal OP */
#define WDCHK1        0     /* 0 : Normal Oper.     1: sysreset */
#define WDCHK0        1     /* 0 : System reset   1: Normal OP */
#define WDSP          7     /* Watchdog prescaler 7 : div 64   */
/*************************************************************/
/*************      SETUP for the SCSR - Register   ************/
#define CLKSRC        0     /* 0 : intern(20MHz)                */
#define LPM           0     /* 0 : Low power mode 0 if idle    */
#define ILLADR        1     /* 1 : clear Illegal Address Flag  */
/*************************************************************/
```

Así quedan definidos los valores que queremos asignarle a cada uno de los bits o secciones que componen el registro empleando la misma nomenclatura que la empleada en su análisis. Recalcar la facilidad que introduce en la programación al no tener que emplear un valor hexadecimal que no aclara a primera vista que opciones se han puesto.

Una vez definido los bits, para escribir el valor completo en el registro se procede a la suma de los valores definidos como números binarios, de tal manera que, para que cada bit se sitúe en su lugar preciso en el registro, cada valor es desplazado hacia la izquierda el número de posiciones que sea necesario. Ello se consigue mediante la estructura "etiqueta<<n", la cual desplaza 'n' posiciones a la izquierda el valor en binario asociado a "etiqueta". Por ejemplo, para los registros que nos ocupa sería:

```
WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                    /* Initialize Watchdog-timer*/

SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR); /* Initialize SCSR */
```

Éste método permite la modificación de la configuración de los registros únicamente con cambiar el valor de los bits en el archivo ".h" o en la zona de programa donde se definen, sin tener que cambiar o trastocar nada en la aplicación principal como ocurría con el primero de los métodos. Más adelante se verá como con operaciones suma de este tipo, OR (|) y operaciones AND (&) se pueden poner a cero, no variar o cambiar ciertas partes de un mismo registro de control

Un caso particular de programación de registros son los mapeados en la memoria de Entrada/Salida, como por ejemplo el registro de control del generador de estados *WSGR*. Éste registro se encuentra en la dirección 0xFFFF de dicha zona de memoria, y para escribir en él hacen falta emplear el mismo tipo de instrucciones que para comunicarse con periféricos externos al DSP. Por defecto, el compilador asume que los valores de direcciones empleados están situados en el mapa de memoria de datos. Por ello, hay que especificar en estos casos que el registro se encuentra en la memoria I/O. De esto se encarga la directiva "ioport", que declara la variable en la memoria I/O. Al igual que antes, para manejar más fácilmente los registros, a esa dirección se le puede asociar una etiqueta mediante la directiva "#define". Para el registro WSGR sería:

```
#define WSGR portFFFF              /* Wait-state generator reg */
ioport unsigned int portFFFF;/*C2xx compiler specific keyword */
```

La estructura portxxxx hace referencia a la dirección xxxx de la memoria I/O como un puerto de comunicaciones, al igual que cuando se desea escribir un dato en un registro de un periférico exterior al DSP. Una vez definida la dirección como I/O se escribe en el registro mediante cualquiera de los métodos descritos anteriormente, como por ejemplo:

```
WSGR = 0x0040;

/*
 bit 15-11     0's:    reserved
 bit 10-9      00:     bus visibility off
 bit 8-6       001:    1 estado de espera para espacio I/O
 bit 5-3       000:    0 estados de espera para espacio datos
 bit 2-0       000:    0 estados para espacio programas
*/
```

Ocasionalmente, pueden encontrarse distintas maneras de configurar este registro. Entre ellas destacan el uso de funciones externas para sacar datos como si de un periférico externo se tratase. Por ejemplo a través de macros del tipo:

```
/*  Macro Expansion Declarations*/
#define _WSGR 0FFFFh /* define _WSGR para uso en ensamblador */

#define STR(x) #x     /* función que da formato a x para ser un
                                    dato para ensamblador */

#define OUTMAC(address,data)  \
 asm("         LDPK    _"STR(data));  \
 asm("         OUT     _"STR(data) "," STR(address))
```

Se define la etiqueta con "_" para indicar que es una variable definida en C y empleada en ensamblador. Se ha definido así una macro "OUTMAC" que saca el dato "data" por la dirección de memoria I/O "address". Esta macro se emplea después para escribir el registro mediante :

```
configdata =    0x0080;     /* 2 esperas para memoria I/O */
OUTMAC( _WSGR, configdata);     /* escribe registro WSGR */
```

En algunos de los ejemplos que se emplearán en este documento, el empleo de una función externa se hace a través de su definición en un archivo a parte en ensamblador llamado "wait.asm", que contiene las siguientes líneas:

```
      .title "wait.asm"
      .globl _out_wsgr
      .sect ".text"
      ; 15.05.2000 Bormann

            NOP
_out_wsgr: SBRK  #1          ; set ARx back to parameter 1
           OUT   *+,0FFFFH     ;I/O output to WSGR-address= FFFF
           RET
```

En ella, la instrucción OUT, saca por la dirección FFFFh el dato que se le haya dado. Ésta función debe ser definida en C mediante la línea:

```
extern _out_wsgr();
```

que define la función como externa al programa en C y con "_" por emplearse en ensamblador. Una vez hecho esto, el dato con la configuración del registro se puede realizar mediante cualquiera de los dos métodos analizados anteriormente, como por ejemplo:

```
/*************   SETUP for the WSGR - Register   **************/
#define BVIS          0     /* 10-9 : 00 Bus visibility OFF    */
#define ISWS          0     /* 8-6 : 000 0 Waitstates for IO   */
#define DSWS          0     /* 5 -3 : 000 0 Waitstates  data   */
#define PSWS          0     /* 2 -0 : 000 0 Waitstaes code     */
/*****************************************************************/

out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                       /* external Function for access  WSGR */
```

Así se construye el dato y se le pasa a la función "_out_wsgr" para que lo escriba en la dirección de memoria I/O correspondiente.

4.4. PUERTOS DE ENTRADA/SALIDA DIGITALES:

Como se analizó anteriormente, el procesador TMS320F243 dispone de 32 pines de entrada/salidas digitales denominadas ***GPIO*** (General Purpose Input Output). Con estos puertos se pueden conectar como entradas interruptores y pulsadores o se pueden generar señales digitales que permitan controlar la activación de relés, lámparas, leds y otros dispositivos controlables por señales digitales (control todo/nada). Trabajan a una tensión de 5V, por lo que serán empleados para controlar el disparo de transistores y el encendido de leds en los ejemplos que se detallarán más adelante.

Los pines de los puertos de Entrada/Salida suelen ser compartidos con otras funciones del DSP, por lo que se encuentran multiplexados. Para el control de estas funciones y de los GPIO, existen 6 registros de 16 bits divididos en dos grupos:

- ***Registros de control de salidas (OCRx):*** Éstos registros controlan el multiplexor que elige entre la función primaria de un pin o la función de Entrada/Salida digital.
- ***Registros de control de datos y direcciones (PxDATDIR):*** Registros que controlan los datos y la dirección en la que salen o entran por cada puerto.

Todos estos registros se encuentran mapeados en la memoria de datos, y aclarar que no existe relación alguna entre los pines GPIO y el espacio I/O del dispositivo.Los registros empleados son los siguientes:

Dirección	Registro	Descripción
7090h	OCRA	Registro de control de salida A
7092h	OCRB	Registro de control de salida B
7098h	PADATDIR	Registro de datos y dirección puerto A
709Ah	PBDATDIR	Registro de datos y dirección puerto B
709Ch	PCDATDIR	Registro de datos y dirección puerto C
709Eh	PDDATDIR	Registro de datos y dirección puerto D

Tabla 4.1. Registros de control de GPIO.

4.4.1. Registro de control de los multiplexores:

A continuación se describen los dos registros que controlan las Entradas/Salidas digitales del DSP: OCRA y OCRB.

4.4.1.1.Registro OCRA:

Éste registro se encuentra mapeado en la dirección de memoria de datos 7090h, y es el encargado de controlar las funciones compartidas de los GPIO correspondientes a los puertos A y B.

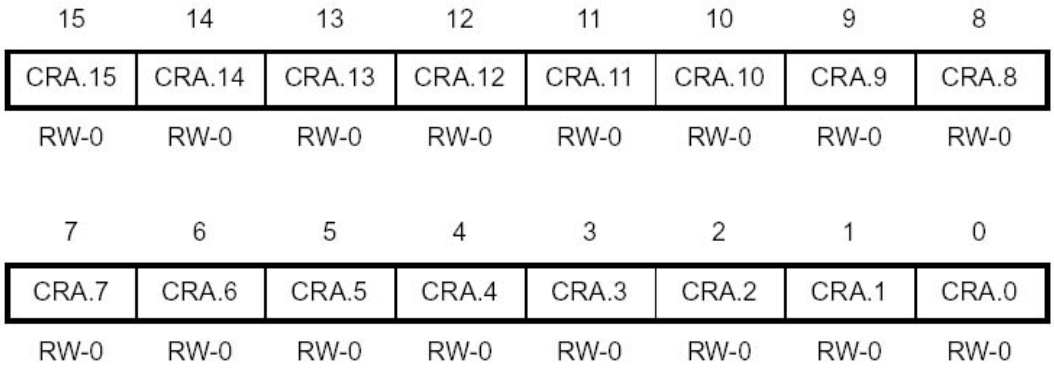


Figura 4.12. Registro de control OCRA (7090h).

Nota: R: acceso de lectura; W: acceso de escritura; -0: valor después de un reset.

En la siguiente tabla se detalla la elección de la función en cada pin según el valor de los bits del registro *OCRA*. El texto resaltado indica que dichas funciones son las seleccionadas después de un reset.

		Función seleccionada	
Bit	**Nombre bit**	**CRA.n=1**	**CRA.n=0**
PUERTO A			
0	CRA.0	SCITXD	***IOPA0***
1	CRA.1	SCIRXD	***IOPA1***
2	CRA.2	XINT1	***IOPA2***
3	CRA.3	CAP1/QEP0	***IOPA3***
4	CRA.4	CAP2/QEP1	***IOPA4***
5	CRA.5	CAP3	***IOPA5***
6	CRA.6	PWM1/CMP1	***IOPA6***
7	CRA.7	PWM2/CMP2	***IOPA7***
PUERTO B			
8	CRA.8	PWM3/CMP3	***IOPB0***
9	CRA.9	PWM4/CMP4	***IOPB1***
10	CRA.10	PWM5/CMP5	***IOPB2***
11	CRA.11	PWM6/CMP6	***IOPB3***
12	CRA.12	T1PWM/T1CMP	***IOPB4***
13	CRA.13	T2PWM/T2CMP	***IOPB5***
14	CRA.14	TDIR	***IOPB6***
15	CRA.15	TCLKIN	***IOPB7***

Tabla 4.2. Configuración Registro OCRA.

De esta manera, poniendo a ‘1’ o ‘0’ cada bit, se selecciona la función para cada uno de los pines de Entrada/Salida. Destacar que cada pin se puede configurar de manera independiente a los demás con la función que se desee.

4.4.1.2.Registro OCRB:

Mediante este registro, situado en la dirección de memoria de datos, 7092h, se controla las funciones compartidas de los pines GPIO correspondientes a los puertos C y D del DSP.

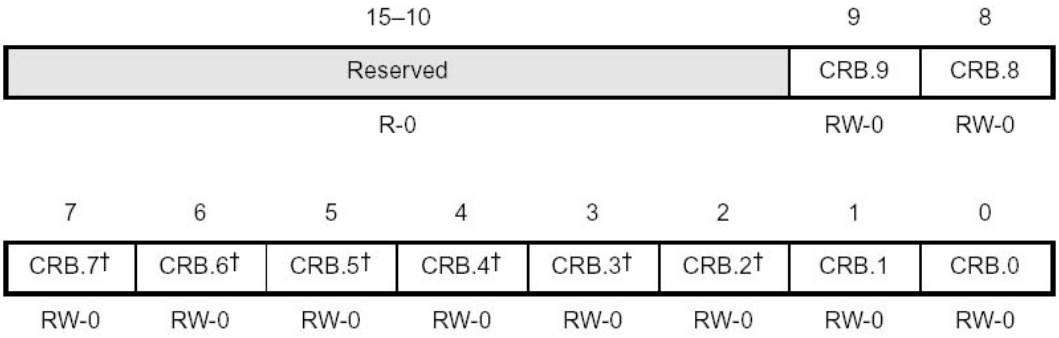


Figura 4.13. Registro de control OCRB (7092h).

Nota: R: acceso de lectura; W: acceso de escritura; -0: valor después de un reset.

En la siguiente tabla se detalla la elección de la función en cada pin según el valor de los bits del registro *OCRB*. El texto resaltado indica que dichas funciones son las seleccionadas después de un reset.

		Función seleccionada	
Bit	**Nombre bit**	**CRB.n=1**	**CRB.n=0**
PUERTO C			
0	CRB.0	IOPC0	***XF***
1	CRB.1	IOPC1	***BIO***
2	CRB.2	SPISIMO	***IOPC2***
3	CRB.3	SPISOMI	***IOPC3***
4	CRB.4	SPICLK	***IOPC4***
5	CRB.5	SPISTE	***IOPC5***
6	CRB.6	CANTX	***IOPC6***
7	CRB.7	CANRX	***IOPC7***
PUERTO D			
8	CRB.8	IOPD0	***CLKOUT***
9	CRB.9	XINT2/ADCSOC	***IOPD1***
10	Reservado		
11	Reservado		
12	Reservado		
13	Reservado		
14	Reservado		
15	Reservado		

Tabla 4.3. Configuración Registro OCRB.

Destacar que 6 de los 8 pines del puerto D están totalmente dedicados como señales GPIO, por lo que no se encuentran multiplexados y no necesitan configurarse en *OCRB*.

4.4.2. Registros de control de datos y dirección:

Existen cuatro registros para el control de los datos y de la dirección en la que se transmiten por cada puerto (tabla 4.1), denominados *PxDATDIR*. A continuación se describe la estructura de cada uno de ellos.

4.4.2.1.Registros PADATDIR, PBDATDIR, PCDATDIR y PDDATDIR:

Éstos registros tienen la misma estructura, por lo que van a ser detallados mediante la descripción de uno de ellos. Están situados en las direcciones de memoria de datos 7098h (*PADATDIR*), 709Ah (*PBDATDIR*), 709Ch (*PCDATDIR*) y 709Eh (*PDDATDIR*). La distribución de uno de ellos, como por ejemplo *PADATDIR*, es la siguiente:

15	14	13	12	11	10	9	8
A7DIR	A6DIR	A5DIR	A4DIR	A3DIR	A2DIR	A1DIR	A0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IOPA7	IOPA6	IOPA5	IOPA4	IOPA3	IOPA2	IOPA1	IOPA0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 4.14. Registro de control de datos y dirección PADATDIR (7098h).

Nota: R: acceso de lectura; W: acceso de escritura; -0: valor después de un reset.

- ***Bits 15-8: AnDIR.*** Indican la dirección en la que va a transmitir los datos el pin corresponidente.

  0 Configura el pin correspondiente como entrada.
  1 Configura el pin correspondiente como salida.

- ***Bits 7-0: IOPAn.*** El significado de estos bits dependen del valor de AnDIR:

  - Si AnDIR=0 (pin es de entrada), entonces:
    - IOPAn=0 se lee el correspondiente pin como a nivel bajo.
    - IOPAn=1 se lee el correspondiente pin como a nivel alto.

  - Si AnDIR=1 (pin es de salida), entonces:
    - IOPAn=0 se establece el correspondiente pin como a nivel bajo.
    - IOPAn=1 se establece el correspondiente pin como a nivel alto.

De igual forma se estructuran los registros *PBDATDIR, PCDATDIR* y *PDDATDIR*: los primeros 8 bits para indicar la dirección de cada uno de los pines y los 8 menos significativos para contener el dato a leer o a transmitir por cada uno de ellos. De esta forma, si se quisieran emplear todos los pines del puerto B como señales de salida (bits

de dirección a '1') para sacar el dato 0x00AB por ejemplo, habría de escribirse en dicho registro:

```
PBDATDIR= 0xFFAB; /*equivale a 1111 1111 1010 1011 en binario */
```

O si, por el contrario, se quiere emplear los pines 0 a 3 del puerto D como entradas y los pines 4 a 7 como salidas para sacar el dato 0x8, debería configurarse el registro como:

```
PDDATDIR= 0xF080; /*equivale a 1111 0000 1000 0000 en binario */
```

Así, para leer el dato que entra en los bits 0 a 3 y guardarlo en una variable llamada "entrada" debe realizarse la siguiente operación:

```
entrada=PDDATDIR & 0x000F;
```

Dicha línea, ha empleado una operación AND (& en lenguaje C) con el número 0x000F (0000 0000 0000 1111 en binario) para poner a cero todos los bits del registro PDDATDIR que no contienen el dato que buscamos y mantener el valor de los bits que sí contienen ese dato (bits 0 a 3). De esta forma se obtiene el valor de las señales de entrada en pines GPIO.

Otras ejemplos habituales de modificación de los registros de los puertos son:

```
PCDATDIR=((y<<4)&0x00F0)|0xF000);/* saca un 4 en bits c3 y c2 */

PDDATDIR= |0xF080; /* al contenido del registro le suma un
            número y guarda el resultado en el mismo registro */
```

4.4.3. Ejemplos de programación:

A continuación se desarrollan una serie de ejemplos para la mejor comprensión del funcionamiento y programación de un dispositivo DSP y de el uso de sus señales GPIO. En cada uno de ellos se hará una introducción explicando cómo se estructura la programación y la funcionalidad que tienen. Éstos ejemplos están diseñados para ponerlos en práctica sobre la placa de pruebas descrita en el apartado 4.2, y además, serán desarrollados mediante las dos herramientas de desarrollo descritos en el capítulo 3: Code Composer y VisSim - Embedded Controls Developer v5.0 for TI C2000. También tienen como objetivo el mostrar cómo se estructura y se desarrolla una aplicación para un DSP en lenguaje C.

4.4.3.1.Ejemplo 1:

Éste primer ejemplo está diseñado para comprender el modo de leer y escribir en los puertos de Entrada/Salida digitales del DSP. Para ello se emplearán los microinterruptores y leds situados en los puertos B y D de la placa de pruebas. El objetivo de esta aplicación es leer el valor de cada una de las señales de entrada del puerto B (que dependerá del estado de los microinterruptores) y mostrar su valor a través de los leds conectados a los pines del puerto D. Éstos leds serán controlados mediante transistores npn de uso general a modo de drivers. Para llevar a cabo esta aplicación, el puerto B debe configurarse con todos sus pines como entradas y el puerto D con todos sus pines como salidas.

Para desarrollar esta aplicación mediante lenguaje C y llevarlo a cabo mediante Code Composer, es necesario el uso de una serie de archivos para generar el proyecto. Algunos de ellos son comunes a todos los ejemplos que se van a desarrollar, por lo que no serán listados repetidamente, como son la librería "rts2xx.lib" incluida en Code Composer y los archivos "regs243.h", "migel.gel", los cuales se detallan en el Anexo 2.

Los archivos necesarios para generar esta aplicación son los siguiente:

- ***F243DIL.c:*** Fichero principal que contiene el código en C para este ejemplo. En él se realizan además las definiciones de los bits y las inicializaciones de los registros del DSP y de las señales de Entrada/Salida. En él aparecen registros de configuración de interrupciones que serán detallados más adelante.

- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.

- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.

- ***F243DIL.cmd:*** Fichero empleado por el enlazador (linker) para la correcta ubicación de variables y constantes en la memoria del DSP.

- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.

- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

A continuación se detalla el código en lenguaje C y ensamblador de cada uno de éstos archivos:

- **Fichero F243DIL.c:**

```
/*********************************************************************/
/* Fichero: F243dil.c                                                */
/* Programa para test de los puertos  B y D de Entrada/Salida        */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz          */
/*********************************************************************/
/* Salidas digitales en puerto D0...D7 , Entradas digitales en puerto B0..B7 */
/* 8 LED's conectados en puerto D0...D7 ; LED-on : 1  LED off : 0    */
/* 8 microinterruptores conectados a GND en el puerto B0..B7         */
/* Los leds muestran el valor de entrada indicado por los microinterrruptores */
/*********************************************************************/

#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */

/********************  SETUP del registro OCRA  *********************/
#define OCRA15          0       /* 0 : IOPB7   1 : TCLKIN          */
#define OCRA14          0       /* 0 : IOPB6   1 : TDIR            */
#define OCRA13          0       /* 0 : IOPB5   1 : T2PWM           */
#define OCRA12          0       /* 0 : IOPB4   1 : T1PWM           */
#define OCRA11          0       /* 0 : IOPB3   1 : PWM6            */
```

```
#define OCRA10                  0       /* 0 : IOPB2   1 : PWM5                 */
#define OCRA9                   0       /* 0 : IOPB1   1 : PWM4                 */
#define OCRA8                   0       /* 0 : IOPB0   1 : PWM3                 */
#define OCRA7                   0       /* 0 : IOPA7   1 : PWM2                 */
#define OCRA6                   0       /* 0 : IOPA6   1 : PWM1                 */
#define OCRA5                   0       /* 0 : IOPA5   1 : CAP3                 */
#define OCRA4                   0       /* 0 : IOPA4   1 :CAP2/QEP2             */
#define OCRA3                   0       /* 0 : IOPA3   1 : CAP1/QEP1            */
#define OCRA2                   0       /* 0 : IOPA2   1 :XINT1                 */
#define OCRA1                   0       /* 0 : IOPA1   1 :SCIRXD                */
#define OCRA0                   0       /* 0 : IOPA0   1 : SCITXD               */
/*********************************************************************/
/*********************    SETUP del registro OCRB    *********************/
#define OCRB9                   0       /* 0 : IOPD1   1 : XINT2/EXTSOC         */
#define OCRB8                   1       /* 0 : CKLKOUT  1 : IOPD0               */
#define OCRB7                   0       /* 0 : IOPC7   1 : CANRX                */
#define OCRB6                   0       /* 0 : IOPC6   1 : CANTX                */
#define OCRB5                   0       /* 0 : IOPC5   1 : SPISTE               */
#define OCRB4                   0       /* 0 : IOPC4   1 : SPICLK               */
#define OCRB3                   0       /* 0 : IOPC3   1 : SPISOMI              */
#define OCRB2                   0       /* 0 : IOPC2   1 : SPISIMO              */
#define OCRB1                   1       /* 0 : BIO     1 : IOPC1                */
#define OCRB0                   1       /* 0 : XF      1 : IOPC0                */
/*********************************************************************/
/*********************     SETUP del WDCR     ************************/
#define WDDIS                   1       /* 0 : Watchdog enabled 1: disabled     */
#define WDCHK2                  1       /* 0 : System reset   1: Normal OP      */
#define WDCHK1                  0       /* 0 : Normal Oper.     1: sys reset    */
#define WDCHK0                  1       /* 0 : System reset   1: Normal OP      */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64        */
/*********************************************************************/
/*********************    SETUP del registro SCSR    *********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                   */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE      */
#define ILLADR                  1       /* 1 : borrar ILLADR                    */
/*********************************************************************/
/************************    SETUP de WSGR    ************************/
#define BVIS            0       /* 10-9 : 00, Bus visibility OFF            */
#define ISWS            0       /* 8 -6 : 000, 0 estados de espera para I/O */
#define DSWS            0       /* 5 -3 : 000, 0 estados de espera para  datos */
#define PSWS            0       /* 2 -0 : 000, 0 estados de espera para código */
/*********************************************************************/

extern _out_wsgr();             /* declaración de función externa para configurar WSGR */

void c_dummy1(void)
{
        while(1);               /* función para atrapar interrupciones espúreas */
}
```

```
/**********************  Programa principal MAIN     **********************/

void main(void)
{
 asm (" setc INTM");          /* Deshabilita todas las interrupciones */
 asm (" clrc SXM");           /* Borra el bit de extensión de signo */
 asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
 asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                                  /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);         /* Inicializa SCSR     */

 out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                        /* Función externa para configurar WSGR */

 OCRB = ((OCRB9<<9)+(OCRB8<<8)+
                (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
                (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                  /* Inicializa OCRB */

 OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                  /* Inicializa OCRA */

 PBDATDIR = 0x0000;              /* Configura IOPB0..B7 como entradas */

 while(1) PDDATDIR = 0xFF00 + (PBDATDIR & 0x00FF);
                        /* bucle sin fin que pone en el puerto D el valor del puerto B */
}
```

Destacar de la última línea las operaciones realizadas para conformar el dato que será escrito en el registro PDDATDIR. Éstas consisten en tomar el valor del registro PBDATDIR y borrar de él los bits dedicados a la dirección de los datos mediante una operación AND con el valor 0x00FF, dejando invariante los bits que contienen al dato. Después, se suma éste valor con 0xFF00 para establecer los bits de dirección como salidas y guardar el resultado en el registro PDDATDIR para sacar el dato por el puerto D. Ésta es una manera muy habitual en programación de intercambiar información entre registros cambiando algunos bits de su configuración.

También se ha empleado uno de los métodos de configuración de registros de control, por el cual se definían uno a uno los bits de cada registro para luego construir el dato mediante suma con desplazamientos.

- **Fichero vectors.asm:**

```
        .title   "vectors.asm"
        .ref     _c_int0,_c_dummy1   ; define punto entrada a código y función dummy
        .sect    ".vectors"

reset:          b       _c_int0
int1:           b       _c_dummy1
int2:           b       _c_dummy1
int3:           b       _c_dummy1
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _c_dummy1
reserved:       b       _c_dummy1
sw_int8:        b       _c_dummy1
sw_int9:        b       _c_dummy1
sw_int10:       b       _c_dummy1
sw_int11:       b       _c_dummy1
sw_int12:       b       _c_dummy1
sw_int13:       b       _c_dummy1
sw_int14:       b       _c_dummy1
sw_int15:       b       _c_dummy1
sw_int16:       b       _c_dummy1
trap:           b       _c_dummy1
nmint:          b       _c_dummy1
emu_trap:       b       _c_dummy1
sw_int20:       b       _c_dummy1
sw_int21:       b       _c_dummy1
sw_int22:       b       _c_dummy1
sw_int23:       b       _c_dummy1
```

- **Fichero F243dil.cmd:**

```
MEMORY
{

   PAGE 0 :   VECS   : origin =    0h , length =   040h   /* VECTORS */
              PROG   : origin =   40h , length = 0FFC0h   /* PROGRAM */

   PAGE 1 :   MMRS   : origin =    0h , length =   060h   /* MMRS    */
                B2   : origin = 0060h , length =   020h   /* DARAM   */
                B0   : origin = 0200h , length =  0100h   /* DARAM   */
                B1   : origin = 0300h , length =  0100h   /* DARAM   */
              DATA   : origin = 8000h , length =  8000h   /* XDM     */

}
```

```
SECTIONS
{
        .text    :{}> PROG PAGE 0
        .cinit   :{}> PROG PAGE 0
        .data    :{}> DATA PAGE 1
        .stack   :{}> DATA PAGE 1
        .bss     :{}> B0   PAGE 1
        .vectors:{}> VECS PAGE 0
}
```

- **Fichero wait.asm:**

```
        .title "wait.asm"
        .globl _out_wsgr
        .sect ".text"
        ; 24.06.1999 Bormann

                NOP
_out_wsgr:      SBRK #1              ; set ARx back to parameter 1
                OUT   *+,0FFFFH      ; I/O-Space-output to WSGR-address = FFFF
                RET
```

Éste mismo ejemplo puede llevarse a cabo mediante el uso del programa VisSim Embedded Controls Developer v5.0 for TI C2000. Para ello basta con situar en la ventana de diagramas un bloque "digital/AnalogInput for C24x" y otro "digital/AnalogOutput for C24x" unidos directamente entre sí (figura 4.15). Ambos bloques se encuentran en la sección "Vissim/DSP" de la barra superior de menús del programa. Si se quiere, puede añadirse un display al diagrama que muestre el valor decimal que se introduce a través de los microinterruptores del puerto B. Para ello sólo es necesario pulsar el icono  en la barra superior de iconos y conectarlo a la salida del bloque "digital/AnalogInput for C24x".

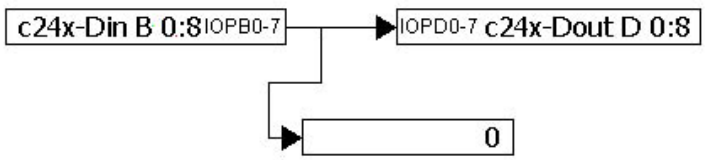


Figura 4.15. Ejemplo 1 con Vissim.

Ambos bloques deben ser configurados para indicar el puerto empleado y la profundidad en bits que van a emplear. Ello se consigue pulsando dos veces sobre cada bloque, con lo que aparece la ventana de configuración (figura 4.16). En ella debe indicarse si el puerto es digital o analógico marcando la opción correspondiente, elegir el puerto empleado (B o D) y la profundidad de bits (Bit Width), que en este caso es de 8 bits (los 8 que tiene cada puerto).

Para ejecutar correctamente éste ejemplo, es necesario configurar las opciones de simulación del diagrama. Ello se conseguía a través de la opción "Simulation Properties..." de la sección "Simulate" del menú superior del programa. En la ventana

de configuración que se abre debe establecerse la casilla “Run in Real Time Mode” y un tiempo de simulación o la opción “Auto Restart" para mantener corriendo el programa durante el tiempo que se desee. Para ejecutar la aplicación basta con pulsar el botón “Go”  .

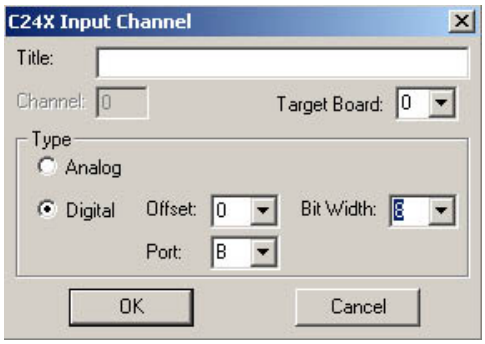


Figura 4.16. Configuración de puertos digitales.

Se comprueba de esta forma la facilidad de uso del entorno de Vissim respecto a la programación en Code Composer. Si fuese necesario Vissim puede generar el código e incluso el archivo compilado “.out”, pero mediante programación con Code Composer puede afinarse más la aplicación en cuanto a tiempos de ejecución y de espacio de memoria se refiere.

4.4.3.2.Ejemplo 2:

Éste segundo ejemplo tiene como objetivo mostrar el empleo de tablas de valores para sacarlos secuencialmente por el puerto D. Exactamente, pretende encender los leds de la placa de pruebas de forma secuencial de izquierda a derecha y de derecha a izquierda, dando lugar a un efecto “Knight Rider” o luz en movimiento. Para ello se empleará únicamente el puerto D de la placa configurado como salida I/O digital. Los archivos necesarios para desarrollar este proyecto son:

- ***LED243C.c:*** Fichero principal que contiene el código en C para este ejemplo. En él se realizan además las definiciones de los bits y las inicializaciones de los registros del DSP y de las señales de Entrada/Salida. En él aparecen registros de configuración de interrupciones que serán detallados más adelante.

- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.

- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.

- ***LED243C.cmd:*** Fichero empleado por el enlazador (linker) para la correcta ubicación de variables y constantes en memoria. El mismo que el caso anterior.

- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.

- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

Éste ejemplo apenas varia respecto del anterior en lo que a archivos se refiere, por lo que solo se detallará el del programa principal. Los demás son los mismos que anteriormente.

- **Fichero LED243C.c:**

```
/*********************************************************************/
/* Fichero: LED243C.c                                                */
/* Programa para test del puerto D de Entrada/Salida                 */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz          */
/*********************************************************************/
/* Salidas digitales en puerto D0...D7                               */
/* 8 LED's conectados en puerto D0...D7 ; LED-on : 1  LED off : 0    */
/* Knight - rider :  1 de 8 LED encendido se desplaza de izquierda a derecha */
/*********************************************************************/

#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */

/*********************    SETUP del registro OCRA    *********************/
#define OCRA15                0       /* 0 : IOPB7   1 : TCLKIN          */
#define OCRA14                0       /* 0 : IOPB6   1 : TDIR            */
#define OCRA13                0       /* 0 : IOPB5   1 : T2PWM           */
#define OCRA12                0       /* 0 : IOPB4   1 : T1PWM           */
#define OCRA11                0       /* 0 : IOPB3   1 : PWM6            */
#define OCRA10                0       /* 0 : IOPB2   1 : PWM5            */
#define OCRA9                 0       /* 0 : IOPB1   1 : PWM4            */
#define OCRA8                 0       /* 0 : IOPB0   1 : PWM3            */
#define OCRA7                 0       /* 0 : IOPA7   1 : PWM2            */
#define OCRA6                 0       /* 0 : IOPA6   1 : PWM1            */
#define OCRA5                 0       /* 0 : IOPA5   1 : CAP3            */
#define OCRA4                 0       /* 0 : IOPA4   1 :CAP2/QEP2        */
#define OCRA3                 0       /* 0 : IOPA3   1 : CAP1/QEP1       */
#define OCRA2                 0       /* 0 : IOPA2   1 :XINT1            */
#define OCRA1                 0       /* 0 : IOPA1   1 :SCIRXD           */
#define OCRA0                 0       /* 0 : IOPA0   1 : SCITXD          */
/*********************************************************************/

/*********************    SETUP del registro OCRB    *********************/
#define OCRB9                 0       /* 0 : IOPD1   1 : XINT2/EXTSOC    */
#define OCRB8                 1       /* 0 : CKLKOUT  1 : IOPD0          */
#define OCRB7                 0       /* 0 : IOPC7   1 : CANRX           */
#define OCRB6                 0       /* 0 : IOPC6   1 : CANTX           */
#define OCRB5                 0       /* 0 : IOPC5   1 : SPISTE          */
#define OCRB4                 0       /* 0 : IOPC4   1 : SPICLK          */
#define OCRB3                 0       /* 0 : IOPC3   1 : SPISOMI         */
#define OCRB2                 0       /* 0 : IOPC2   1 : SPISIMO         */
#define OCRB1                 1       /* 0 : BIO     1 : IOPC1           */
#define OCRB0                 1       /* 0 : XF      1 : IOPC0           */
/*********************************************************************/
```

```
/*********************      SETUP del WDCR      ************************/
#define WDDIS                   1       /* 0 : Watchdog enabled 1: disabled      */
#define WDCHK2                  1       /* 0 : System reset   1: Normal OP       */
#define WDCHK1                  0       /* 0 : Normal Oper.      1: sys reset    */
#define WDCHK0                  1       /* 0 : System reset   1: Normal OP       */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64         */
/*****************************************************************************/

/*********************    SETUP del registro SCSR    *********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                    */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE       */
#define ILLADR                  1       /* 1 : borrar ILLADR                     */
/*****************************************************************************/

/************************    SETUP de WSGR    ************************/
#define BVIS            0       /* 10-9 : 00, Bus visibility OFF                */
#define ISWS            0       /* 8 -6 : 000, 0 estados de espera para I/O     */
#define DSWS            0       /* 5 -3 : 000, 0 estados de espera para  datos  */
#define PSWS            0       /* 2 -0 : 000, 0 estados de espera para código  */
/*****************************************************************************/

unsigned int LED[8]={0xFF01,0xFF02,0xFF04,0xFF08,0xFF10,0xFF20,
                                                        0xFF40,0xFF80};
                                        /* tabla de 8 datos para puerto D */
unsigned char n;
extern _out_wsgr();          /* declaración de función externa para configurar WSGR */

void wait(void)                         /* Función que genera un retraso de tiempo */
{
        unsigned int i;
        for(i=0;i<65000;i++);
}

void c_dummy1(void)
{
        while(1);               /* función para atrapar interrupciones espúreas */
}

/*********************   Programa principal MAIN    *********************/

void main(void)
{
 asm (" setc INTM");            /* Deshabilita todas las interrupciones */
 asm (" clrc SXM");             /* Borra el bit de extensión de signo */
 asm (" clrc OVM");             /* Borra bit de modo de desbordamiento */
 asm (" clrc CNF");             /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                        /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);        /* Inicializa SCSR     */
```

```
out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                        /* Función externa para configurar WSGR */

OCRB = ((OCRB9<<9)+(OCRB8<<8)+
              (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
              (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                    /* Inicializa OCRB */

OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
      (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
      (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
      (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                    /* Inicializa OCRA */

while(1){
       for(n=0;n<14;n++){
              if(n<7) PDDATDIR = LED[n];      /* Bucle sin fin donde va sacando
              else PDDATDIR=LED[14-n];              por el puerto D los valores
              wait();                                              de la tabla */
              }
       }
}
```

Para dar la impresión de una luz en movimiento, se ha empleado una tabla de valores en las que se tienen almacenados los 8 valores que tomará PDDATDIR. Cada valor es de la forma "0xFFnn", de manera que se emplea el puerto D como salida digital, y siendo "nn" el dato a sacar por dicho puerto, que no es más que un '1' en los distintos bits según el led que se quiera encender. Dependiendo del orden en el que se vayan tomando estos valores, la luz se moverá en un sentido o en otro, para lo cual emplea una variable auxiliar 'n' que varía de 0 a 13, que son los 14 posibles estados del movimiento. Mientras n<7, se toman los valores del primero al último, y cuando n>7 se toman del último al primero. De esta manera se crea el efecto visual. Para que le de tiempo al ojo humano a detectar el encendido de cada led, se ha definido una función "wait" que introduce un retraso de tiempo entre la toma de un valor y otro. Ésta función únicamente toma el valor 65000 y lo va decrementando hasta llegar a cero. Si se quiere tener el movimiento a mayor velocidad, basta con disminuir este valor, como por ejemplo a 32000, que da lugar a un movimiento con la mitad de frecuencia. Por lo demás, el resto del programa es idéntico al ejemplo 1.

4.5. TRATAMIENTO DE INTERRUPCIONES:

Las interrupciones son una de las características más importantes de cualquier tipo de procesador, debido a que permiten suspender de manera temporal el programa principal para responder a un determinado evento, ya sea interno o externo. De esta manera se evita la necesidad de comprobar periódicamente si se han producido determinados eventos en los periféricos o componentes externos al sistema. Esto mejora el tiempo de respuesta del procesador y disminuye su uso. Habitualmente las interrupciones son generadas por dispositivos que necesitan dar o recibir datos del procesador, como por ejemplo convertidores A/D. Por otro lado, las interrupciones pueden emplearse también como señales que informan al procesador de que se ha dado un cierto evento en el sistema. Así, el procesador detiene la aplicación principal y comienza a ejecutar la porción de código que atiende a dicho evento..

El procesador TMS320F243 soporta seis interrupciones no emascarables agrupadas en seis niveles (INT1 a INT6) y una no enmascarable (*NMI*). Debido a el gran número de periféricos de que dispone dicho procesador, éstos seis niveles de interrupción deben ser compartidos por todos los dispositivos internos y externos, por lo que deberán ser jerarquizados, como se verá a continuación. Éste modelo de procesadores admite tres tipos de interrupción:

- **Interrupciones generadas por software** mediante instrucciones INTR, TRAP, NMI, etc.
- **Interrupciones generadas por hardware** procedentes de pines externos o de los periféricos internos del procesador.
- **Reset hardware o software** procedentes de pines externos o del watchdog. Son las de mayor prioridad.

Para que el procesador atienda una determinada interrupción, deben darse unas determinadas condiciones en los bits de una serie de registros de control de interrupciones, los cuales actúan a modo de interruptores que conectan la interrupción con la CPU (figura 4.16). De esta forma, existe un bit de habilitación global de las interrupciones *INTM* que se encuentra en el registro de estado *ST0*. Si *INTM* = 0, se habilitan globalmente todas las interrupciones enmascarables del sistema, y si *INTM* = 1 quedan todas deshabilitadas (recordar que, al ser el bit de un registro de estado, es puesto a '1' mediante la instrucción "*SETC INTM*" en ensamblador, y puesto a '0' mediante la instrucción "*CLRC INTM*"). Además de éste requisito, para que se atienda una interrupción ésta debe haberse requerido previamente. Para ello, cuando se produce cualquier interrupción en uno de los seis niveles (INT1 a INT6), se produce un cambio en el indicador (flag) correspondiente a dicha interrupción, el cual se almacena en el registro indicador de interrupción *IFR* si está habilitada esa interrupción. Éste indicador permanece invariable hasta que es borrado por el procesador al mandar la señal de reconocimiento de interrupción o vía software. Éste bit debe ser borrado para permitir el reconocimiento de posteriores interrupciones. Para darse el reconocimiento en *IFR*, debe estar habilitada dicha interrupción en el registro de máscaras de interrupciones *IMR*. Éste registro permite enmascarar las interrupciones INT1 a INT6. Si en el bit correspondiente hay un '1', esa interrupción está habilitada, lo contrario deshabilita la interrupción. La interrupción se atiende si, además de permitirlo el bit de habilitación global *INTM*, está habilitada en registro *IMR* y el indicador correspondiente de *IFR* está activo.

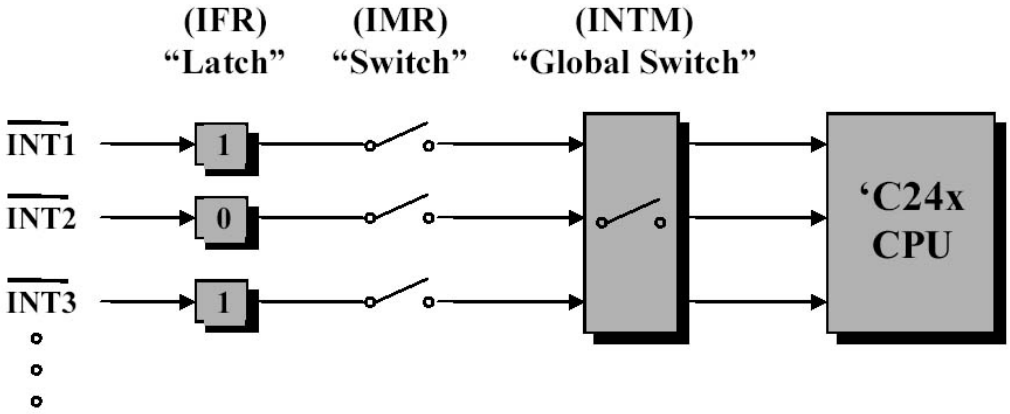


Figura 4.17. Registros de control de interrupciones.

Éste es el modo en el que se comprueba si la interrupción producida debe atenderse, pero existe un problema previo a éste: la coordinación del gran número de interrupciones que pueden producir los numerosos periféricos para atenderlos en sólo seis niveles (INT1 a INT6). Para ello, el DSP dispone de una unidad especial que se encarga de organizar y jerarquizar todas las interrupciones en esos seis niveles, denominándose Unidad de Expansión de Interrupciones Periféricas (PIE).

4.5.1. Unidad de Expansión de Interrupciones Periféricas (PIE):

Para manejar directamente todas las peticiones de interrupción de los periféricos, el procesador TMS320F243 dispone de una unidad especial que genera un arbitraje, establece un nivel de prioridad y efectúa el control de las peticiones de interrupción, agrupándolas para actuar sobre las seis interrupciones disponibles en la CPU (INT1 a INT6). Éste módulo especial se denomina Unidad de Expansión de Interrupciones Periféricas (PIE), cuya misión es gestionar todas las interrupciones hardware y software con un nivel determinado de prioridad. Dicho módulo, agrupa en cada una de las seis interrupciones de la CPU las distintas interrupciones internas o externas que pueden requerir al procesador (figura 4.17), pero se hace necesaria la existencia de una lógica de arbitraje y gestión de interrupciones que indique a la CPU qué periférico, de los agrupados en cada nivel, es el que he requerido la interrupción. Ello se consigue a través de los siguientes registros:

- **Registro de petición de periférico *PIRQRn*:** Detecta y registra cualquier interrupción generada desde los periféricos, indicando si la interrupción está pendiente de tratamiento.
- **Registro de reconocimiento de interrupción de periférico *PIACKRn*:** Confirma qué interrupción ha sido reconocida de entre las que están pendientes e indica al periférico correspondiente el reconocimiento de la interrupción que solicitó.
- **Registro de vector de interrupción de periféricos *PIVR*:** Contiene el vector correspondiente a la interrupción (INT1 a INT6) de la que se ha producido el reconocimiento.

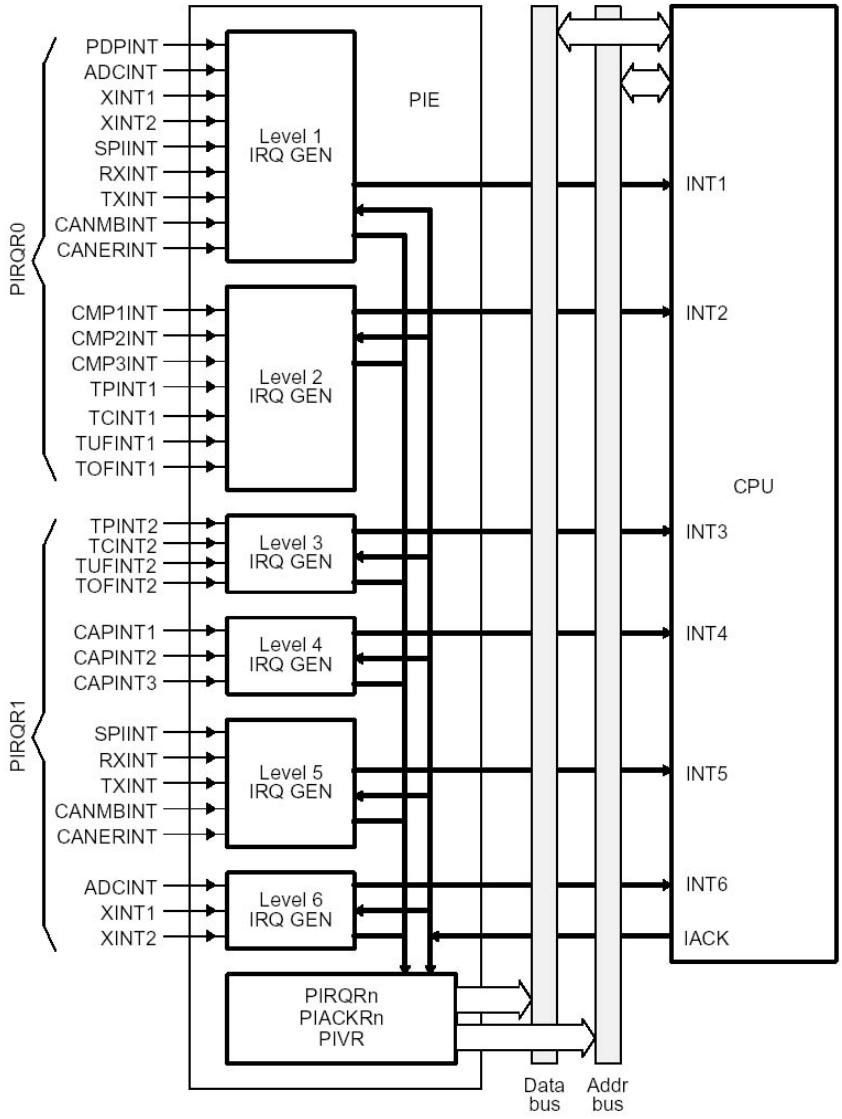


Figura 4.18. Organización de interrupciones en la PIE.

El proceso por el que se determina la procedencia de la interrupción es el siguiente:

Cada petición de interrupción generada por un periférico precisa de su correspondiente señal de reconocimiento por parte de la CPU. Cuando esto ocurre, la CPU coloca en el bus de direcciones de programa un valor que corresponde al del nivel de interrupción que ha sido reconocida, de forma que pueda encontrarse el vector de interrupción dentro de la memoria de programa, ya que cada interrupción INTn tiene su propio vector en una dirección de memoria de programa particular (tabla 4.18). Por ejemplo, si se confirma la interrupción INT3, el vector que se carga es el 0006h. Éstos vectores se emplean para iniciar la denominada "rutina de servicio de interrupciones (GISR)" correspondiente al nivel de prioridad INTn que se va a tratar. Ésta rutina se encarga de leer el registro *PIVR* (el cual ha sido cargado por el controlador PIE al ver el reconocimiento para indicar el periférico cuya interrupción se ha reconocido) y con el que, después de salvar el contexto necesario, genera un vector a la denominada "rutina de servicio específico de la interrupción generada (SISR)", la cual efectuará las acciones que correspondan al periférico que generó la interrupción según se haya programado la aplicación.

Nombre de la interrupción	Prioridad	Dirección vector de interrupción	PIVR	Descripción de la interrupción
Reset	1	0000h	N/A	Reset desde pin externo o del watchdog
Reservado	2	0026h	N/A	Trap del emulador
NMI	3	0024h	N/A	Interrupt. Solo software
PDPINT	4	INT1 0002h	0020h	Pin de interrupción por protección de potencia
ADCINT	5		0004h	Interrupt. por ADC (AP)
XINT1	6		0001h	Pines externos de interruptción (AP)
XINT2	7		0011h	
SPIINT	8		0005h	Pines externos SPI (AP)
RXINT	9		0006h	Recepción del SCI (AP)
TXINT	10		0007h	Transmisión SCI (AP)
CANMBINT	11		0040h	Mailbox del CAN (AP)
CANERINT	12		0041h	Error del CAN (AP)
CMP1INT	13	INT2 0004h	0021h	Comparación 1 (EV)
CMP2INT	14		0022h	Comparación 2 (EV)
CMP3INT	15		0023h	Comparación 3 (EV)
TPINT1	16		0027h	Periodo Timer 1 (EV)
TCINT1	17		0028h	Compara Timer 1 (EV)
TUFINT1	18		0029h	Underflow Timer 1 (EV)
TOFINT1	19		002Ah	Overflow Timer 1 (EV)
TPINT2	20	INT3 0006h	002Bh	Periodo Timer 2 (EV)
TCINT2	21		002Ch	Compara Timer 2 (EV)
TUFINT2	22		002Dh	Underflow Timer 2 (EV)
TOFINT2	23		002Eh	Overflow Timer 2 (EV)
CAPINT1	24	INT4 0008h	0033h	Captura 1 (EV)
CAPINT2	25		0034h	Captura 2 (EV)
CAPINT3	26		0035h	Captura 3 (EV)
SPIINT	27	INT5 000Ah	0005h	Pines externos del SPI (BP)
RXINT	28		0006h	Recepción del SCI (BP)
TXINT	29		0007h	Transmisión SCI (BP)
CANMBINT	30		0040h	Mailox del CAN (BP)
CANERINT	31		0041h	Error del CAN (BP)
ADCINT	32	INT6 000Ch	0004h	Interrupt. por ADC (BP)
XINT1	33		0001h	Pines externos interrupción (Baja prioridad)
XINT2	34		0011h	
Reservado	-	000Eh	N/A	Análisis de interrupción
TRAP	N/A	0022h	N/A	Instrucción TRAP
Phantom	N/A	N/A	0000h	Interrupción del vector de phantom

Figura 4.19. Tabla general de interrupciones del TMS320F243.

Nota: AP = Alta prioridad; BP = Baja prioridad.

En la tabla anterior se detallan las distintas interrupciones que pueden producir los periféricos del DSP, ordenados por nivel de prioridad e indicando los vectores de interrupción y *PIVR*. Cada uno de ellos tiene su correspondiente bit asociado en los registros de control de periféricos *PIRQ0* y *PIRQ1*, los cuales se ponen a '1' cuando el periférico asociado hace una petición de interrupción. Si dicha interrupción está habilitada, el controlador PIE pone a '1' el correspondiente bit del registro *IFR* para indicar que existe una interrupción pendiente del reconocimiento.

Para que una interrupción pueda ser tratada correctamente, en la programación de la aplicación deben tenerse en cuenta dos puntos:

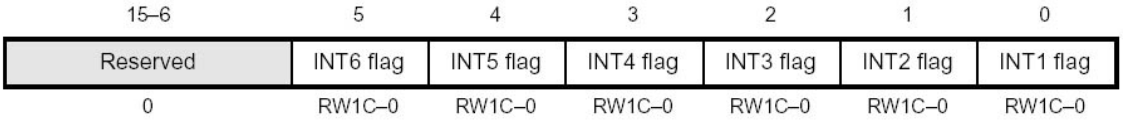
1. Asignar en el archivo "vectors.asm" un nombre con el que identificar la rutina de tratamiento de cada interrupción, como por ejemplo: INT3: B ADC_rut

2. Definir una función que se corresponda con esa rutina de interrupción que contenga el código oportuno para el tratamiento de la interrupción, y en la que inicialmente se compruebe el valor de *PIVR* para comprobar si ha provocado la interrupción el periférico al que está asociado dicha rutina. En caso de no ser así, no debe ejecutarse la rutina de interrupción. Al final de ésta rutina deben borrarse los flag indicadores de interrupción del periférico (si este dispone de ellos, como el Event Manager) para permitir nuevas interrupciones del mismo.

Cuando se programa en lenguaje C, la librería "rts2xx.lib" aporta todo el código necesario para salvar y reponer el contexto cuando se atiende a una interrupción, entendiendo por contexto al estado del DSP antes de atender la rutina, como son los registros de estado, FIFO, etc. En el apartado dedicado el Gestor de Eventos se detallará un ejemplo de utilización de interrupciones.

4.5.2. Registro de indicadores de interrupciones (IFR):

Éste registro de 16 bits está mapeado en la dirección de memoria de datos 0006h y en él se contienen los indicadores de las interrupciones activadas en la CPU. Con ellos se identifica el nivel de petición de interrupción ocurrido (INT1 a INT6) y borrar las interrupciones pendientes. El indicador correspondiente a un nivel de interrupción se pone a '1' cuando existe una petición de interrupción por parte de un periférico (bit en PIRQn activo) y si está habilitada dicha interrupción. Cuando la CPU establece el reconocimiento de la misma, éste bit es borrado para permitir nuevas interrupciones en ese nivel. También puede borrarse en un reset o por software escribiendo un '1' en el bit correspondiente. Una práctica habitual para borrar todas las interrupciones pendientes es escribir el contenido actual del registro *IFR* sobre sí mismo



15–6	5	4	3	2	1	0
Reserved	INT6 flag	INT5 flag	INT4 flag	INT3 flag	INT2 flag	INT1 flag
0	RW1C–0	RW1C–0	RW1C–0	RW1C–0	RW1C–0	RW1C–0

Figura 4.20. Registro de indicadores de interrupciones IFR (0006h).

Nota: 0 = siempre se lee como ceros; R = acceso de lectura; W1C = escribir un 1 en ese bit para ponerlo a cero; -0 = valor después de un reset.

- ***Bits 15-6: Reservados.*** Siempre se leen como ceros.

- ***Bit 5: INT6.*** Indicador de interrupciones asociadas al nivel INT6 de interrupciones.
  0 No existe interrupción pendiente.
  1 Al menos existe una petición de interrupción pendiente en el nivel INT6.

- ***Bit 4: INT5.*** Indicador de interrupciones asociadas al nivel INT5 de interrupciones.
  0 No existe interrupción pendiente.
  1 Al menos existe una petición de interrupción pendiente en el nivel INT5.

- ***Bit 3: INT4.*** Indicador de interrupciones asociadas al nivel INT4 de interrupciones.
  0 No existe interrupción pendiente.
  1 Al menos existe una petición de interrupción pendiente en el nivel INT4.

- ***Bit 2: INT3.*** Indicador de interrupciones asociadas al nivel INT3 de interrupciones.
  0 No existe interrupción pendiente.
  1 Al menos existe una petición de interrupción pendiente en el nivel INT3.

- ***Bit 1: INT2.*** Indicador de interrupciones asociadas al nivel INT2 de interrupciones.
  0 No existe interrupción pendiente.
  1 Al menos existe una petición de interrupción pendiente en el nivel INT2.

- ***Bit 0: INT1***. Indicador de interrupciones asociadas al nivel INT1 de interrupciones.
  0 No existe interrupción pendiente.
  1 Al menos existe una petición de interrupción pendiente en el nivel INT1.

4.5.3. Registro de máscara de interrupciones (IMR):

Registro de 16 bits mapeado en la dirección de memoria de datos 0004h, encargado de habilitar el nivel de las interrupciones enmascarables INT1 a INT6. Cada nivel se activa poniendo a '1' el bit correspondiente. Leyendo éste registro se comprueban qué interrupciones están habilitadas, las cuales se atenderán sólo si lo permite el bit INTM. Los bits del registro *IMR* no se borran con un reset.

15–6	5	4	3	2	1	0
Reserved	INT6 mask	INT5 mask	INT4 mask	INT3 mask	INT2 mask	INT1 mask
0	RW	RW	RW	RW	RW	RW

Figura 4.21. Registro de máscara de interrupciones IMR (0004h).

Nota: 0 = siempre se lee como ceros; R = acceso de lectura; W = acceso de escritura.

- ***Bits 15-6: Reservados.*** Siempre se leen como ceros.

- ***Bit 5: INT6.*** Máscara de interrupciones del nivel INT6 de interrupciones.
  0 Nivel INT6 inhabilitado.
  1 Nivel INT6 habilitado.

- ***Bit 4: INT5.*** Máscara de interrupciones del nivel INT5 de interrupciones.
  0 Nivel INT5 inhabilitado.
  1 Nivel INT5 habilitado.

- ***Bit 3: INT4.*** Máscara de interrupciones del nivel INT4 de interrupciones.
  0 Nivel INT4 inhabilitado.
  1 Nivel INT4 habilitado.

- ***Bit 2: INT3.*** Máscara de interrupciones del nivel INT3 de interrupciones.
  0 Nivel INT3 inhabilitado.
  1 Nivel INT3 habilitado.

- ***Bit 1: INT2.*** Máscara de interrupciones del nivel INT2 de interrupciones.
  0 Nivel INT2 inhabilitado.
  1 Nivel INT2 habilitado.

- ***Bit 0: INT1.*** Máscara de interrupciones del nivel INT1 de interrupciones.
  0 Nivel INT1 inhabilitado.
  1 Nivel INT1 habilitado.

4.5.4. Registro del vector de interrupciones de periféricos (PIVR):

El registro de 16 bits *PIVR* se encuentra mapeado en la dirección de memoria de datos 701Eh, siendo de solo lectura. Es el encargado de cargar el vector de interrupción de la petición de más alta prioridad asociada con la interrupción de la CPU (INTx) que ha sido reconocida durante el ciclo de reconocimiento del periférico. El significado de sus bits es el siguiente:

15	14	13	12	11	10	9	8
V15	V14	V13	V12	V11	V10	V9	V8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	V0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 4.22. Registro del vector de interrupción de periféricos PIVR (701Eh).

Nota: -0 = valor después de un reset; R = acceso de lectura.

- ***Bits 15-0: V15-V0.*** Registro que contiene el vector de interrupción correspondiente al periférico que ha solicitado la interrupción y es de más reciente reconocimientos.

4.5.5. Ejemplos de programación:

A la hora de programar una aplicación en la que se empleen interrupciones es recomendable seguir una serie de pasos, los cuales sirven de orientación al programador y preparan la aplicación para correcto funcionamiento de las mismas. Anteriormente, se hicieron breves comentarios sobre las modificaciones necesarias en un código para emplear interrupciones. Con el fin de aclarar estos conceptos, a continuación serán descritos los ejemplos 1 y 2, anteriormente utilizados, en los que se han introducido las modificaciones necesarias para emplear interrupciones.

Antes de comenzar con los ejemplos, conviene detallar en qué consisten las modificaciones introducidas:

1- Un primer cambio se da en el archivo "vectors.asm". En éste archivo deben asociarse las distintas interrupciones con los nombres de las rutinas que se tienen que ejecutar. Por ejemplo, si la interrupción 1 es empleada para ejecutar la rutina "ADC_ISR" escrita en un archivo en lenguaje C, debe primero definirse esa variable en el archivo "vectors.asm" y luego asociarla con INT1 como sigue:

```
   .global    _ADCISR    ;define la variable
              .
              .
INT1:    B    _ADCISR    ; asocia INT1 con la rutina
              .
              .
```

2- La siguiente modificación se dará en los archivos en los que se definen las direcciones de los registros y sus bits. Esto es porque se van a emplear nuevos registros en la programación que antes no fueron utilizados, como son: *IMR, IFR, EVIFRA, EVIMRA*, etc. Dichos registros son los encargados de habilitar las interrupciones e indicar las que se han producido. En próximos capítulos se detallarán más a fondo los registros *EVIFRx* y *EVIMRx*, los cuales enmascaran y contienen los flag de las interrupciones del gestor de eventos. Por ello deben incluirse sus direcciones y bits en los archivos que contengan estos datos: include ".h", archivos ".c", etc.

3- Los cambio más importantes se dan en el archivo del programa principal. Éste debe modificar ligeramente su estructura para incluir nuevas líneas de código. Una estructura sencilla y que es la empleada en los ejemplos que se detallan a continuación es la siguiente:

- Definición de registros y bits.
- Definición de diversas funciones empleadas.
- Definición de la rutina de interrupción. En ella debe comprobarse el origen de la interrupción, ya que por el nivel de interrupción 1, por ejemplo, pueden producirse interrupciones de distintos periféricos. Por ello es conveniente comprobar el registro que contiene el vector de interrupción de periféricos *PIVR* para

comprobar que la interrupción ha sido producida por el periférico para el que está preparada ésta rutina (tabla 4.18). Los periféricos que pueden solicitar interrupciones, suelen contener un registro en el que exista un flag que indique la petición de interrupción. Éste indicador debe ser borrado al final de la rutina para permitir próximas peticiones de interrupción por parte del periférico. En lenguaje C presentan la siguiente estructura:

```
Interrupt void ADC_ISR (void)  /* define función interrupción */
{
     if((PIVR-0x0004)==0)       /* Comprueba origen de la
     {                                          interrupción */
          .
          .          /* rutina de interrupción */
          .
          .

      ADCTRL1 |= 0x0100;   /* borra flag del ADC y guarda
     }                          resultado en el mismo registro */
}
```

- En la función principal main, las primeras líneas suelen ser de iniciación del entorno y de registros. A la hora de emplear interrupciones deben añadirse nuevas líneas, las cuales se encargan de deshabilitar y habilitar globalmente las interrupciones enmascarables (*INTM*), enmascarar los niveles de interrupciones (*IMR*) o borrar las posibles interrupciones en espera de procesos anteriores (*IFR*). Para que no existan problemas en éste bloque de iniciación, lo primero que debe hacerse es deshabilitar globalmente todas las interrupciones enmascarables del sistema, de forma que no se produzca ninguna mientras se inician los registros.

```
void main (void)           /* función principal */
{
     asm("   setc INTM");  /* deshabilita todas las
                                        interrupciones */
```

Además, en la iniciación de registros deben incluirse la configuración de los registros que controlan las interrupciones, como son: *IMR, IFR, EVIFRA, EVIMRA*, etc.

```
     EVIFRA=0xFFFF;        /* borra flags del grupo A de EV */

     EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)
             +(CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));

             /* Enmascara interrupciones del grupo A del EV */

     IFR=0xFFFF;      /* borra los posibles flags activos */

     IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+
                                                    (INT1));

                      /* Enmascara interrupciones del sistema */
```

Por último, y antes del bucle principal del programa, deben habilitarse nuevamente todas las interrupciones enmascarables del sistema.

```
asm(“ clrc INTM”);   /* habilita todas las interrupciones */

while(1)
{.....
```

Ésta estructura puede comprobarse en los siguiente ejemplos, lo cual no quiere decir que pueda programarse la aplicación de un modo distinto, pero a modo de ejemplo a nivel académico aclara muchos conceptos sobre el emplea de interrupciones y de su configuración.

4.5.5.1.Ejemplo 3:

Éste ejemplo se corresponde con el ejemplo 1 descrito anteriormente para el empleo de los puertos de Entrada/Salida digitales del procesador. En él se han realizado modificaciones para que exista una rutina de interrupción que se encargue de mostrar el estado de los microinterruptores del puerto B en los leds conectados al puerto D, lo cual se hacía anteriormente mediante un bucle infinito. Ahora, con las modificaciones empleadas, se procede a generar una interrupción cada 0.2 segundos que se encarga de leer y escribir ambos puertos. Para conseguir este objetivo, se emplea uno de los temporizadores de uso general que dispone el gestor de eventos del DSP (GPT1), el cual funciona a modo de iniciador de la lectura y escritura de los puertos al generar una interrupción cuando el contador llega a un determinado valor. Para introducir un tiempo de retraso de 1 segundo entre acceso y acceso a los puertos, la rutina espera a que se den 5 interrupciones antes de acceder a los puertos (5*0.2s = 1 s). Más adelante se entrará en detalles sobre éste dispositivo.

Los archivos necesarios para generar esta aplicación son prácticamente los mismos que los empleados en el ejemplo 1, salvo el archivo principal y de vectores que han sufrido modificaciones, por lo que sólo se detallarán éstos últimos:

- ***F243DIL2.c:*** Fichero principal que contiene el código en C para este ejemplo. En él se añaden registros de configuración de interrupciones y del temporizador de usos general GPT1

- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.

- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.

- ***F243DIL.cmd:*** Fichero empleado por el enlazador (linker) para la correcta ubicación de variables y constantes en la memoria del DSP.

- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.

- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

- **Fichero F243dil2.c:**

```
/*****************************************************************/
/* Fichero: F243dil2.c                                           */
/* Programa para test de los puertos  B y D de Entrada/Salida    */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz      */
/*****************************************************************/
/* Salidas digitales en puerto D0...D7 , Entradas digitales en puerto B0..B7 */
/* 8 LED's conectados en puerto D0...D7 ; LED-on : 1  LED off : 0 */
/* 8 microinterruptores conectados a GND en el puerto B0..B7     */
/* Los leds muestran el valor de entrada indicado por los microinterrruptores */
/* Timer GPT1 produce una interrupción cada 0.2 segundos         */
/* La rutina de interrupción espera a 5 interrupciones para acceder a  los puertos */
/*****************************************************************/

#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */

/*********************  SETUP del registro OCRA  *********************/
#define OCRA15              0       /* 0 : IOPB7   1 : TCLKIN        */
#define OCRA14              0       /* 0 : IOPB6   1 : TDIR          */
#define OCRA13              0       /* 0 : IOPB5   1 : T2PWM         */
#define OCRA12              0       /* 0 : IOPB4   1 : T1PWM         */
#define OCRA11              0       /* 0 : IOPB3   1 : PWM6          */
#define OCRA10              0       /* 0 : IOPB2   1 : PWM5          */
#define OCRA9               0       /* 0 : IOPB1   1 : PWM4          */
#define OCRA8               0       /* 0 : IOPB0   1 : PWM3          */
#define OCRA7               0       /* 0 : IOPA7   1 : PWM2          */
#define OCRA6               0       /* 0 : IOPA6   1 : PWM1          */
#define OCRA5               0       /* 0 : IOPA5   1 : CAP3          */
#define OCRA4               0       /* 0 : IOPA4   1 :CAP2/QEP2      */
#define OCRA3               0       /* 0 : IOPA3   1 : CAP1/QEP1     */
#define OCRA2               0       /* 0 : IOPA2   1 :XINT1          */
#define OCRA1               0       /* 0 : IOPA1   1 :SCIRXD         */
#define OCRA0               0       /* 0 : IOPA0   1 : SCITXD        */
/*****************************************************************/

/*********************  SETUP del registro OCRB  *********************/
#define OCRB9               0       /* 0 : IOPD1   1 : XINT2/EXTSOC  */
#define OCRB8               1       /* 0 : CKLKOUT  1 : IOPD0        */
#define OCRB7               0       /* 0 : IOPC7   1 : CANRX         */
#define OCRB6               0       /* 0 : IOPC6   1 : CANTX         */
#define OCRB5               0       /* 0 : IOPC5   1 : SPISTE        */
#define OCRB4               0       /* 0 : IOPC4   1 : SPICLK        */
#define OCRB3               0       /* 0 : IOPC3   1 : SPISOMI       */
#define OCRB2               0       /* 0 : IOPC2   1 : SPISIMO       */
#define OCRB1               1       /* 0 : BIO     1 : IOPC1         */
#define OCRB0               1       /* 0 : XF      1 : IOPC0         */
/*****************************************************************/
```

```
/*********************      SETUP del WDCR      ************************/
#define WDDIS                   1       /* 0 : Watchdog enabled 1: disabled      */
#define WDCHK2                  1       /* 0 : System reset   1: Normal OP       */
#define WDCHK1                  0       /* 0 : Normal Oper.      1: sys reset    */
#define WDCHK0                  1       /* 0 : System reset   1: Normal OP       */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64         */
/*****************************************************************************/

/*********************     SETUP del registro SCSR     *********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                    */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE       */
#define ILLADR                  1       /* 1 : borrar ILLADR                     */
/*****************************************************************************/

/************************    SETUP de WSGR    ***************************/
#define BVIS            0       /* 10-9 : 00, Bus visibility OFF             */
#define ISWS            0       /* 8 -6 : 000, 0 estados de espera para I/O  */
#define DSWS            0       /* 5 -3 : 000, 0 estados de espera para  datos */
#define PSWS            0       /* 2 -0 : 000, 0 estados de espera para código */
/*****************************************************************************/

/*******************      SETUP del registro GPTCON     *********************/
#define GPTCON_T2TOADC          0
                        /* 10-9  : T2TOADC = 00 : GPT2 no inicia ADC         */
#define GPTCON_T1TOADC          0
                        /*  8-7  : T1TOADC = 00 : GPT1 no inicia ADC         */
#define GPTCON_TCOMPOE          0
        /*   6   : TCOMPOE = 0  : deshabilitadas las salidas de comparación del GPT */
#define GPTCON_T2PIN            0
                    /*  3-2  : T2PIN = 00 : Pol. de salida GPT2 = forzada nivel bajo */
#define GPTCON_T1PIN            0
                    /*  1-0  : T1PIN = 00 : Pol. de salida GPT1 = forzada nivel bajo */
/*****************************************************************************/

/*********************      SETUP del registro T1CON     *********************/
#define T1CON_FREESOFT          0
                    /* 15-14 FREE, SOFT : 00 stop en emulación JTAG suspendido   */
#define T1CON_TMODE             2
                    /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T1CON_TPS               7
            /* 10-8  : TPS2-0     : 111  prescaler del reloj de entrada CPUCLK/128  */
#define T1CON_TENABLE           1       /*  6    : TENABLE    : 1 habilita GPT1     */
#define T1CON_TCLKS             0
                                /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T1CON_TCLD              1
                    /* 3-2   : TCLD1,0    : 01 Recarga del timer 1 al llegar a 0 o a T */
#define T1CON_TECMPR            1
            /* 1     :  TECMPR     : 1 habilita operación de comparación del timer    */
/*****************************************************************************/
```

```
/********************    SETUP del registro EVIMRA    *********************/
#define T1OFINT                 0  /* 10 : Timer 1 overflow interrupt          */
#define T1UFINT                 0  /*  9 : Timer 1 underflow interrupt         */
#define T1CINT                  0  /*  8 : Timer 1 compare interrupt           */
#define T1PINT                  1  /*  7 : Timer 1 period interrupt            */
#defineCMP3INT                  0  /*  3 : Compare 3 interrupt                 */
#define CMP2INT                 0  /*  2 : Compare 2 interrupt                 */
#define CMP1INT                 0  /*  1 : Compare 1 interrupt                 */
#define PDPINT                  0  /*  0 : Power Drive Protect Interrupt       */
/*****************************************************************************/

/********************    SETUP del registro EVIMRB    *********************/
#define T2OFINT                 0  /*  3 : Timer 2 overflow interrupt          */
#define T2UFINT                 0  /*  2 : Timer 2 underflow interrupt         */
#define T2CINT                  0  /*  1 : Timer 2 compare interrupt           */
#define T2PINT                  0  /*  0 : Timer 2 period interrupt            */
/*****************************************************************************/

/********************    SETUP del registro EVIMRC    *********************/
#define CAP3INT                 0  /*  2 : Capture Unit 3 interrupt            */
#define CAP2INT                 0  /*  1 : Capture Unit 2 Interrupt            */
#define CAP1INT                 0  /*  0 : Capture unit 1 interrupt            */
/*****************************************************************************/

/***********************    SETUP del registro IMR    ***********************/
#define INT6                  0  /*  5 : Level INT6  enmascarada               */
#define INT5                  0  /*  4 : Level INT5  enmascarada               */
#define INT4                  0  /*  3 : Level INT4  enmascarada               */
#define INT3                  0  /*  2 : Level INT3  enmascarada               */
#define INT2                  1  /*  1 : Level INT2  no enmascarada            */
#define INT1                  0  /*  0 : Level INT1  enmascarada               */
/*****************************************************************************/

#define PERIOD 31250        /* T1 PERIODO = 50ns * 128 * 31250 = 0.2s*/

interrupt void T1PER_ISR(void);

extern _out_wsgr();           /* declaración de función externa para configurar WSGR */

void c_dummy1(void)
{
        while(1);               /* función para atrapar interrupciones espúreas */
}

interrupt void T1PER_ISR(void)              /* Rutina de interrupción */
{
 static unsigned char i=0;
 if ((PIVR-0x0027)==0)    /* Verifica origen de interrupción-Nr. 0027 = T1PINT*/
 {
   i++;                   /* cuenta el número de interrupciones        */
```

```
   if(i>=5){                        /* 5 interrupciones hacen: 5*0.2s = 1.0 s */
                PDDATDIR= 0xFF00 + (PBDATDIR & 0x00FF);
                i=0;                                        /* Actualiza los leds */
                }
   EVIFRA=(T1PINT<<7);          /* borra flag de interrupción del temporizador */
 }
}

/*********************    Programa principal MAIN  **********************/

void main(void)
{
asm (" setc INTM");          /* Deshabilita todas las interrupciones */
asm (" clrc SXM");           /* Borra el bit de extensión de signo */
asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                                  /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);        /* Inicializa SCSR     */

out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                    /* Función externa para configurar WSGR */

 OCRB = ((OCRB9<<9)+(OCRB8<<8)+
                (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
                (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                      /* Inicializa OCRB */

 OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                      /* Inicializa OCRA */

GPTCON=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
            (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                                    /* Inicia el registro de control del temporizador */

PDDATDIR = 0xFF00;           /* Apaga los leds del puerto D */
PBDATDIR = 0x0000;           /* Configura puerto B como entradas */
T1PR = PERIOD;               /* Configura el periodo del temporizador */
T1CNT= 0x0000;               /* Valor inicial del contador */

 T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+(T1CON_TPS<<8)+
               (T1CON_TCLKS<<4)+(T1CON_TCLD<<2)+(T1CON_TECMPR<<1));
                                    /* Configura el registro de control del temporizador 1 */
```

```
 EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
                        (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                                        /* Registro de máscaras del grupo A del EV  */

 EVIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                                        /* Registro de máscaras del grupo B del EV  */

 EVIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                                        /* Registro de máscaras del grupo C del EV  */

 EVIFRA=0xFFFF;      /* Registro de flags del grupo A del EV  */
 EVIFRB=0xFFFF;      /* Registro de flags del grupo B del EV  */
 EVIFRC=0xFFFF;      /* Registro de flags del grupo C del EV  */

 IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                                                /* Registro de máscaras de interrupción */

 IFR=0xFFFF;             /* Borra todas las interrupciones pendientes */

 asm (" clrc INTM");             /* Habilita todas las interrupciones */

 T1CON=T1CON+(T1CON_TENABLE<<6);                 /* Habilita el GPT1*/

 while(1);               /* Bucle sin fin. Todo lo realiza la rutina de interrupción */
}
```

Éste archivo responde a la estructura indicada anteriormente. De esta forma, se añade en la rutina de interrupción la comprobación del registro *PIVR* para determinar si la interrupción dada en el nivel INT2 se corresponde con la producida por el temporizador 1 al llegar a su periodo (*PIVR* = 0x0027 según tabla 4.18). Destacar que el inicio del temporizador se hace en última instancia cuando es habilitado éste dispositivo, de manera que empiece a contar desde el valor inicial asignado justo después de habilitar las interrupciones. Los registros *EVIMRx* y *EVIFRx* tienen la misma función que los registros *IMR* y *IFR*, pero respecto de los dispositivos que dispone el gestor de eventos.

- **Fichero vectors.asm:**

```
        .title    "vectors.asm"
        .ref      _c_int0,_c_dummy1,_T1PER_ISR
        .sect     ".vectors"

reset:          b       _c_int0
int1:           b       _c_dummy1
int2:           b       _T1PER_ISR
int3:           b       _c_dummy1
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _c_dummy1
reserved:       b       _c_dummy1
```

```
sw_int8:    b    _c_dummy1
sw_int9:    b    _c_dummy1
sw_int10:   b    _c_dummy1
sw_int11:   b    _c_dummy1
sw_int12:   b    _c_dummy1
sw_int13:   b    _c_dummy1
sw_int14:   b    _c_dummy1
sw_int15:   b    _c_dummy1
sw_int16:   b    _c_dummy1
trap:       b    _c_dummy1
nmint:      b    _c_dummy1
emu_trap:   b    _c_dummy1
sw_int20:   b    _c_dummy1
sw_int21:   b    _c_dummy1
sw_int22:   b    _c_dummy1
sw_int23:   b    _c_dummy1
```

La tabla de vectores de interrupción ha sido modificada para atender la interrupción INT2 mediante la rutina "T1PER_ISR" definida en el archivo principal. Los demás parámetros permanecen invariantes frente al ejemplo 1.

4.5.5.2.Ejemplo 4:

Al igual que el ejemplo anterior, éste está basado en uno de los ejemplos anteriormente descritos. En concreto es una variación del ejemplo 2 en el que se introduce una rutina de interrupción que se encarga de encender los leds en la secuencia programada y de generar el tiempo de espera necesario entre el encendido de un led y el siguiente, consiguiéndose así un efecto de luz en movimiento. El temporizador de uso general GPT2 se encarga de solicitar una interrupción cada 0.1 segundos, los cuales sirven de tiempo de retraso para el encendido de los leds. Para ello son necesarias las modificaciones analizadas anteriormente, por lo que sólo serán detallados los archivos que se ven modificados.

- ***LED243C2.c:*** Fichero principal que contiene el código en C para este ejemplo. Se introducen aquí las modificaciones necesarias para el empleo de rutinas de interrupción.

- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.

- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.

- ***LED243C.cmd:*** Fichero empleado por el enlazador (linker).

- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.

- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

- **Fichero LED243C2.c:**

```
/*********************************************************************/
/* Fichero: LED243C2.c                                               */
/* Programa para test del puerto D de Entrada/Salida                 */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz          */
/*********************************************************************/
/* Salidas digitales en puerto D0...D7                               */
/* 8 LED's conectados en puerto D0...D7 ; LED-on : 1  LED off : 0    */
/* Knight - rider :  1 de 8 LED encendido se desplaza de izquierda a derecha */
/* Tiempo de retraso de 0.1 segundos producido por la interrupción del Timer 2 */
/*********************************************************************/

#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */

/*********************   SETUP del registro OCRA   *********************/
#define OCRA15                0       /* 0 : IOPB7   1 : TCLKIN      */
#define OCRA14                0       /* 0 : IOPB6   1 : TDIR        */
#define OCRA13                0       /* 0 : IOPB5   1 : T2PWM       */
#define OCRA12                0       /* 0 : IOPB4   1 : T1PWM       */
#define OCRA11                0       /* 0 : IOPB3   1 : PWM6        */
#define OCRA10                0       /* 0 : IOPB2   1 : PWM5        */
#define OCRA9                 0       /* 0 : IOPB1   1 : PWM4        */
#define OCRA8                 0       /* 0 : IOPB0   1 : PWM3        */
#define OCRA7                 0       /* 0 : IOPA7   1 : PWM2        */
#define OCRA6                 0       /* 0 : IOPA6   1 : PWM1        */
#define OCRA5                 0       /* 0 : IOPA5   1 : CAP3        */
#define OCRA4                 0       /* 0 : IOPA4   1 :CAP2/QEP2    */
#define OCRA3                 0       /* 0 : IOPA3   1 : CAP1/QEP1   */
#define OCRA2                 0       /* 0 : IOPA2   1 :XINT1        */
#define OCRA1                 0       /* 0 : IOPA1   1 :SCIRXD       */
#define OCRA0                 0       /* 0 : IOPA0   1 : SCITXD      */
/*********************************************************************/
/*********************   SETUP del registro OCRB   *********************/
#define OCRB9                 0       /* 0 : IOPD1   1 : XINT2/EXTSOC */
#define OCRB8                 1       /* 0 : CKLKOUT  1 : IOPD0      */
#define OCRB7                 0       /* 0 : IOPC7   1 : CANRX       */
#define OCRB6                 0       /* 0 : IOPC6   1 : CANTX       */
#define OCRB5                 0       /* 0 : IOPC5   1 : SPISTE      */
#define OCRB4                 0       /* 0 : IOPC4   1 : SPICLK      */
#define OCRB3                 0       /* 0 : IOPC3   1 : SPISOMI     */
#define OCRB2                 0       /* 0 : IOPC2   1 : SPISIMO     */
#define OCRB1                 1       /* 0 : BIO     1 : IOPC1       */
#define OCRB0                 1       /* 0 : XF      1 : IOPC0       */
/*********************************************************************/
/*********************   SETUP del WDCR   *********************/
#define WDDIS                 1       /* 0 : Watchdog enabled 1: disabled */
#define WDCHK2                1       /* 0 : System reset   1: Normal OP */
#define WDCHK1                0       /* 0 : Normal Oper.     1: sys reset */
#define WDCHK0                1       /* 0 : System reset   1: Normal OP */
#define WDSP                  7       /* Watchdog prescaler 7 : div 64 */
/*********************************************************************/
```

```
/*********************     SETUP del registro SCSR    *********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE   */
#define ILLADR                  1       /* 1 : borrar ILLADR                 */
/*****************************************************************************/
/************************     SETUP de WSGR    ************************/
#define BVIS            0       /* 10-9 : 00, Bus visibility OFF             */
#define ISWS            0       /* 8 -6 : 000, 0 estados de espera para I/O  */
#define DSWS            0       /* 5 -3 : 000, 0 estados de espera para  datos */
#define PSWS            0       /* 2 -0 : 000, 0 estados de espera para código */
/*****************************************************************************/
/*******************      SETUP del registro GPTCON    *********************/
#define GPTCON_T2TOADC      0
                            /* 10-9  : T2TOADC = 00 : GPT2 no inicia ADC     */
#define GPTCON_T1TOADC      0
                            /*  8-7  : T1TOADC = 00 : GPT1 no inicia ADC     */
#define GPTCON_TCOMPOE      1
            /*   6   : TCOMPOE = 1  : habilitadas las salidas de comparación del GPT */
#define GPTCON_T2PIN        1
                        /*  3-2  : T2PIN = 01 : Pol. de salida GPT2 = activa nivel bajo */
#define GPTCON_T1PIN        0
                        /*  1-0  : T1PIN = 00 : Pol. de salida GPT1 = forzada nivel bajo */
/*****************************************************************************/
/*********************      SETUP del registro T2CON    *********************/
#define T2CON_FREESOFT      0
                    /* 15-14 FREE, SOFT : 00 stop en emulación JTAG suspendido   */
#define T2CON_TMODE         2
                        /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T2CON_TPS           6
                /* 10-8  : TPS2-0     : 110  prescaler del reloj de entrada CPUCLK/64  */
#define T2CON_TSWT1         0   /*  7: TSWT1: 0 usa su propio bit TENABLE */
#define T2CON_TENABLE       1       /*  6     : TENABLE    : 1 habilita GPT2      */
#define T2CON_TCLKS         0
                                /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T2CON_TCLD          1
                        /* 3-2   : TCLD1,0    : 01 Recarga del timer 2 al llegar a 0 o a T */
#define T2CON_TECMPR        1
            /* 1     :  TECMPR     : 1 habilita operación de comparación del timer     */
#define T2CON_SELT1PR       0
                        /* 0        :  SELT1PR     :0       use su propio registro de periodo  */
/*****************************************************************************/
/*******************     SETUP del registro EVIMRA    *********************/
#define T1OFINT             0  /* 10 : Timer 1 overflow interrupt               */
#define T1UFINT             0  /*  9 : Timer 1 underflow interrupt              */
#define T1CINT              0  /*  8 : Timer 1 compare interrupt                */
#define T1PINT              0  /*  7 : Timer 1 period interrupt                 */
#defineCMP3INT              0  /*  3 : Compare 3 interrupt                      */
#define CMP2INT             0  /*  2 : Compare 2 interrupt                      */
#define CMP1INT             0  /*  1 : Compare 1 interrupt                      */
#define PDPINT              0  /*  0 : Power Drive Protect Interrupt            */
/*****************************************************************************/
```

```
/*******************    SETUP del registro EVIMRB    *********************/
#define T2OFINT                 0  /*  3 : Timer 2 overflow interrupt          */
#define T2UFINT                 0  /*  2 : Timer 2 underflow interrupt         */
#define T2CINT                  0  /*  1 : Timer 2 compare interrupt           */
#define T2PINT                  1  /*  0 : Timer 2 period interrupt            */
/*****************************************************************************/

/*******************    SETUP del registro EVIMRC    *********************/
#define CAP3INT                 0  /*  2 : Capture Unit 3 interrupt            */
#define CAP2INT                 0  /*  1 : Capture Unit 2 Interrupt            */
#define CAP1INT                 0  /*  0 : Capture unit 1 interrupt            */
/*****************************************************************************/

/***********************    SETUP del registro IMR    ***********************/
#define INT6                  0  /*  5 : Level INT6  enmascarada               */
#define INT5                  0  /*  4 : Level INT5  enmascarada               */
#define INT4                  0  /*  3 : Level INT4  enmascarada               */
#define INT3                  1  /*  2 : Level INT3  no enmascarada            */
#define INT2                  0  /*  1 : Level INT2  enmascarada               */
#define INT1                  0  /*  0 : Level INT1  enmascarada               */
/*****************************************************************************/

#define PERIOD 31250        /* Para que el periodo sea T2PERIOD = 0,1s se tienen que
                                                T2PR = 0,1s / ( 50ns * T2CON_TPS ) */

unsigned int LED[8]={0xFF01,0xFF02,0xFF04,0xFF08,
                                        0xFF10,0xFF20,0xFF40,0xFF80};
                                        /* Tabla de valores para el encendido de los leds */

interrupt void T2PER_ISR(void);             /* declaración de rutina de interrupción */

extern _out_wsgr();           /* declaración de función externa para configurar WSGR */

void c_dummy1(void)
{
        while(1);               /* función para atrapar interrupciones espúreas */
}

interrupt void T2PER_ISR(void)              /* Rutina de interrupción */
{
 static unsigned char i=0;
 if ((PIVR-0x002B)==0)      /* Verifica origen de interrupción -Nr. 002B = T2PINT*/
 {
  if(i<7)PDDATDIR=LED[i++];             /* Secuencia de dencendido de los leds */
  else PDDATDIR=LED[14-i++];
  if (i>=14) i=0;
  EVIFRB=T2PINT;                    /* Borra el flag de interrupción del timer 2 */
 }
}
```

```
/**********************    Programa principal MAIN  ***********************/

void main(void)
{
asm (" setc INTM");          /* Deshabilita todas las interrupciones */
asm (" clrc SXM");           /* Borra el bit de extensión de signo */
asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                          /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);        /* Inicializa SCSR     */

out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                    /* Función externa para configurar WSGR */

 OCRB = ((OCRB9<<9)+(OCRB8<<8)+
              (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
              (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                  /* Inicializa OCRB */

 OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                  /* Inicializa OCRA */

GPTCON=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
          (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                                  /* Inicia el registro de control del temporizador */

PDDATDIR = 0xFF00;           /* Apaga los leds del puerto D */
T2PR = PERIOD;               /* Inicia periodo del timer 2*/
T2CMPR = PERIOD/2;           /* Valor del comparador para duty cycle 1:1*/
T2CNT=0x0000;               /* Valor inicial del contador del T2 */

T2CON=((T2CON_FREESOFT<<14)+(T2CON_TMODE<<11)+(T2CON_TPS<<8)+
               (T2CON_TSWT1<<7)+(T2CON_TCLKS<<4)+(T2CON_TCLD<<2)+
                                    (T2CON_TECMPR<<1)+T2CON_SELT1PR);
                                /* Configura el registro de control del temporizador 2 */

EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
                     (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                                    /* Registro de máscaras del grupo A del EV  */

 EVIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                                    /* Registro de máscaras del grupo B del EV  */

 EVIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                                    /* Registro de máscaras del grupo C del EV  */
```

```
 EVIFRA=0xFFFF;      /* Registro de flags del grupo A del EV  */

 EVIFRB=0xFFFF;      /* Registro de flags del grupo B del EV  */

 EVIFRC=0xFFFF;      /* Registro de flags del grupo C del EV  */

 IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                                                  /* Registro de máscaras de interrupción */

 IFR=0xFFFF;             /* Borra todas las interrupciones pendientes */

 asm (" clrc INTM");             /* Habilita todas las interrupciones */

 T2CON=T2CON+(T2CON_TENABLE<<6);                 /* Habilita el GPT2*/

 while(1);               /* Bucle sin fin. Todo lo realiza la rutina de interrupción */
}
```

En este caso, la rutina de interrupción es la encargada de ir seleccionando los valores de la tabla de datos con la que crear el efecto de luz en movimiento. Ello lo realiza cada 0.1 segundos, cuando es llamada esta rutina por producirse una interrupción por periodo del timer 2. En ella se comprueba inicialmente que el vector *PIVR* = 0x0028 correspondiente con el valor asociado a dicho tipo de interrupción (tabla 4.18). Al final de la misma se borra el flag de interrupción del timer 2 para permitir próximas interrupciones de éste dispositivo. Éste ejemplo añade la utilización del timer 2 para generar una onda cuadrada de duty cycle = 50% en el pin de salida correspondiente. Esto se verá con más detalle en apartados posteriores.

- **Fichero vectors.asm:**

```
        .title    "vectors.asm"
        .ref      _c_int0,_c_dummy1,_T2PER_ISR
        .sect     ".vectors"


reset:          b       _c_int0
int1:           b       _c_dummy1
int2:           b       _c_dummy1
int3:           b       _T2PER_ISR
int4:           b       _c_dummy1
int5:           b       _c_dummy1
int6:           b       _c_dummy1
reserved:       b       _c_dummy1
sw_int8:        b       _c_dummy1
sw_int9:        b       _c_dummy1
sw_int10:       b       _c_dummy1
sw_int11:       b       _c_dummy1
```

```
sw_int12:   b     _c_dummy1
sw_int13:   b     _c_dummy1
sw_int14:   b     _c_dummy1
sw_int15:   b     _c_dummy1
sw_int16:   b     _c_dummy1
trap:       b     _c_dummy1
nmint:      b     _c_dummy1
emu_trap:   b     _c_dummy1
sw_int20:   b     _c_dummy1
sw_int21:   b     _c_dummy1
sw_int22:   b     _c_dummy1
sw_int23:   b     _c_dummy1
```

Éste archivo únicamente recoge las rutinas que son asociadas a los diferentes niveles de interrupción del procesador. En particular, asocia al nivel INT3 de interrupciones la rutina "T2PER_ISR" definida en el archivo principal del programa y que debe ser declarada al principio de éste archivo. Al igual que en los casos anteriores, los demás niveles se asocian con la rutina "c_dummy1" para retener interrupciones no deseadas, excepto la interrrupción por "Reset" que siempre queda asociada al punto de entrada del código "c_int0".

4.6. GESTOR DE EVENTOS O EVENT MANAGER (EV):

El procesador TMS320F243, dispone de un módulo especial con dispositivos dedicados al control digital denominado Gestor de eventos o Event Manager (EV). Éste proporciona un gran número de funciones y característica que para el caso que nos ocupa, el control de motores, lo hacen especialmente importante. Como se comentó anteriormente, los pines de Entradas/Salidas digitales estaban compartidos con otras funciones, las cuales son las salidas y entradas del EV, con lo que existirán distintos registros que configuren una función u otra. Ésta característica se repite en todos los modelos de la serie C2000 de Texas Instruments, aunque hay que destacar que entre dichos modelos se encuentran algunas diferencias en cuanto al EV se refiere, tales como mayor número de registros y prestaciones en unos que en otros, distinta modo de empleo de alguno de los periféricos, etc. Para hacer compatibles las nomenclaturas de los distintos modelos, en la referencia técnica del TMS320F243, se denomina al Gestor de Eventos EV2. Ello se debe a que en otros modelos de la serie C2000 existen dos gestores de eventos.

El Gestor de Eventos del procesador TMS320F243 contiene los siguientes bloques funcionales (figura 4.22):

- Dos temporizadores de uso general o GPT.
- Tres unidades de comparación.
- Circuitos de generación de señales PWM, las cuales incluyen la modulación PWM Space Vector (SVPWM), generación de tiempos muertos (dead-band) y selección de lógica de salida.
- Tres unidades de captura.
- Circuito de pulsos de cuadratura de encoder o QEP.
- Interrupciones lógicas por parte de éstos dispositivos.

Éstos Bloques se repiten en los distintos modelos de la serie C2000, pero hay una serie de diferencias que son detalladas en le referencia técnica del TMS320F243.

Como se muestra en la figura 4.22, el Gestor de Eventos dispone de una serie de pines de entras y salidas, los cual se pueden agrupar en los siguientes grupos (tabla *):

- o Dos salidas de los dos temporizadores de propósito general: T1CMP/T1PWM y T2CMP/T2PWM.
- o Seis salidas de las unidades de comparación: PWM1 a PWM6.
- o Tres pines de captura: CAP1/QEP1, CAP2/QEP1 y CAP3.
- o Pin de entrada de reloj externo: TCLKIN.
- o Pin de entrada de dirección de cuenta externa del temporizador: TDIR.

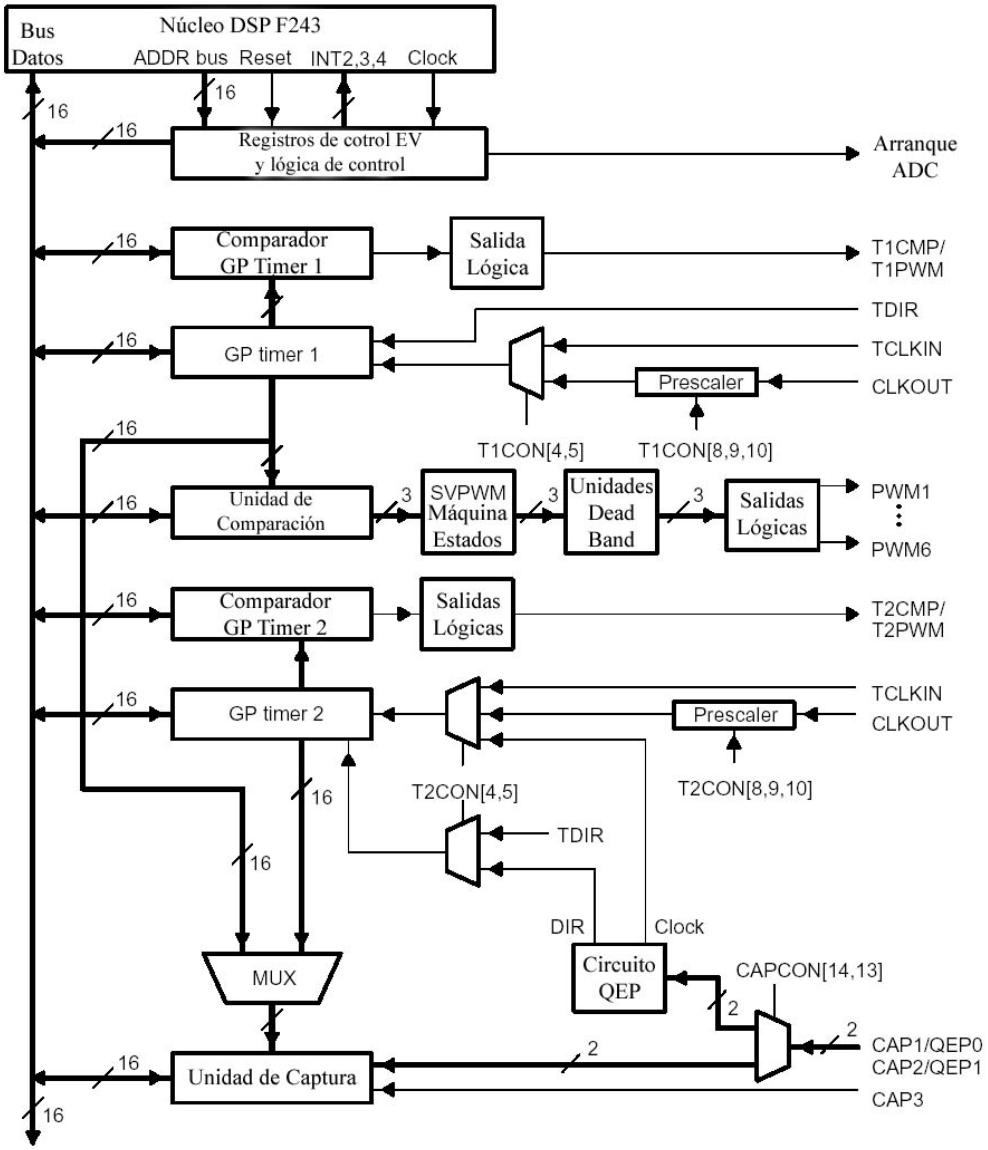


Figura 4.23. Diagrama de bloques del EV.

Pines del Módulo Gestor de Eventos (EV)	
CAP1/QEP0	Entrada 1 de la unidas de captura, entrada 0 del QEP.
CAP2/QEP1	Entrada 2 de la unidas de captura, entrada 1 del QEP.
CAP3	Entrada 3 de la unidas de captura.
PWM1	Salida digital 1 de la unidad de comparación 1.
PWM2	Salida digital 2 de la unidad de comparación 1.
PWM3	Salida digital 1 de la unidad de comparación 2.
PWM4	Salida digital 2 de la unidad de comparación 2.
PWM5	Salida digital 1 de la unidad de comparación 3.
PWM6	Salida digital 2 de la unidad de comparación 3.
T1CMP/T1PWM	Comparación Timer 1, Salida PWM1.
T2CMP/T2PWM	Comparación Timer 2, Salida PWM2.
TCLKIN	Pin de entrada de reloj externo para temporizadores del EV.
TDIR	Pin entrada de dirección del temporizador externo del EV.

Tabla 4.4. Pines externos del Gestor de Eventos (EV).

Para el control y configuración de los distintos dispositivos que componen el Gestor de Eventos, existen una serie de registros de control de cada uno de ellos, con los que se puede configurar tanto su funcionamiento (modo de conteo, periodo de señales, modos de captura, registros de almacenamiento de datos, etc.) como la gestión de sus interrupciones a la CPU. Dichos registros de encuentran mapeados en la memoria de datos, y se detallan en la siguiente tabla:

Registros de control de los Temporizadores en el EV		
Dirección	Registro	Nombre
0x7400	GPTCON	Registro de control general de temporizadores
0x7401	T1CNT	Registro contador del Timer1
0x7402	T1CMPR	Registro comparador del Timer1
0x7403	T1PR	Registro de periodo del Timer1
0x7404	T1CON	Registro de control del Timer 1
0x7405	T2CNT	Registro contador del Timer2
0x7406	T2CMPR	Registro comparador del Timer2
0x7407	T2PR	Registro de periodo del Timer2
0x7408	T2CON	Registro de control del Timer 2
Registro de control de los Comparadores en el EV		
0x7411	COMCON	Registro de control del comparador
0x7413	ACTR	Registro de acción del comparador
0x7415	DBTCON	Registro de control de tiempos muertos
0x7417	CMPR1	Registro de comparación 1
0x7418	CMPR2	Registro de comparación 2
0x7419	CMPR3	Registro de comparación 3
Registros de Captura en el EV		
0x7420	CAPCON	Registro de control de unidad de captura
0x7422	CAPFIFO	Registro de estado de FIFO de captura
0x7423	CAP1FIFO	FIFO1 de captura de dos niveles
0x7424	CAP2FIFO	FIFO2 de captura de dos niveles
0x7425	CAP3FIFO	FIFO3 de captura de dos niveles
0x7427	CAP1BOT	Registro del segundo nivel de captura en FIFO1
0x7428	CAP2BOT	Registro del segundo nivel de captura en FIFO2
0x7429	CAP3BOT	Registro del segundo nivel de captura en FIFO3
Registro de Interrupciones en el EV		
0x742C	EVAIMRA	Registro de máscaras de interrupción del grupo A
0x742D	EVAIMRB	Registro de máscaras de interrupción del grupo B
0x742E	EVAIMRC	Registro de máscaras de interrupción del grupo C
0x742F	EVAIFRA	Registro de flags de interrupción del grupo A
0x7430	EVAIFRB	Registro de flags de interrupción del grupo B
0x7431	EVAIFRC	Registro de flags de interrupción del grupo C

Tabla 4.5. Registros y direcciones del Gestor de Eventos (EV).

4.6.1. Temporizadores de propósito general:

Existen dos temporizadores de uso general en el Gestor de Eventos del procesador TMS320F243. éstos pueden usarse como bases de tiempo independientes en aplicaciones como:

- Generación de periodos de muestreo en un sistema de control.
- Proveer una base de tiempos para el circuito de codificación de pulsaos de cuadratura o QEP y para las unidades de captura.
- Generación de señales PWM utilizando las unidades de comparación.

Para ello, cada temporizador dispone de una serie de registros que controlan la actuación del temporizador (figura 4.23). A continuación se describen la función de los distintos registros en el proceso del temporizador (para x = 1,2):

- ***TxCNT:*** **Registro contador del temporizador.** Éste registro almacena el valor actual de la cuenta y lo incrementa o decrementa según la dirección programada para la misma. En él se pueden leer y escribir datos.
- ***TxCMPR:*** **Registro comparador del temporizador.** Registros que contiene el valor con el que se va a comparar el valor del contador correspondiente de manera continuada. Se puede emplear para indicar el duty cycle o ciclo de trabajo útil de una onda PWM cuando se emplea el temporizador para ello. Puede ser modificado en cualquier momento, ya que dispone de un registro auxiliar oculto en el que se precarga inicialmente el valor escrito (shadowed). Éste valor pasa al registro comparador cuando ocurre el evento configurado para ello en el registro de control (*TxCON*), como es que la cuente llegue a cero, o al periodo, etc. Cuando se da una comparación se producen los siguientes eventos:
  - Se produce una transición en la salida de comparación TxCMP/TxPWM asociada, un ciclo de reloj después de alcanzar la comparación.
  - Se activa el correpondiente indicador de interrupción un ciclo de reloj después del evento.
  - Si está habilitado, se indica el inicio de una conversión en el ADC en el mismo momento que se establece a ‘1’ el indicador de interrupción.
- ***TxPR:*** **Registro de periodo del temporizador.** Indica el periodo de la cuenta que lleva a cabo el temporizador. Dispone de registro auxiliar de precarga de datos para poder modificarlo en cualquier momento. La actualización del valor se produce cuando el registro contador *TxCNT* llega a cero.
- ***TxCON:*** **Registro de control del temporizador**, en el que se configura el modo de funcionamiento del mismo.
- ***GPTCON:*** **Registro de control global de los dos temporizadores**. En él se indican las acciones que se deben llevar a cabo por loe temporizadores cuando se dan ciertos eventos..

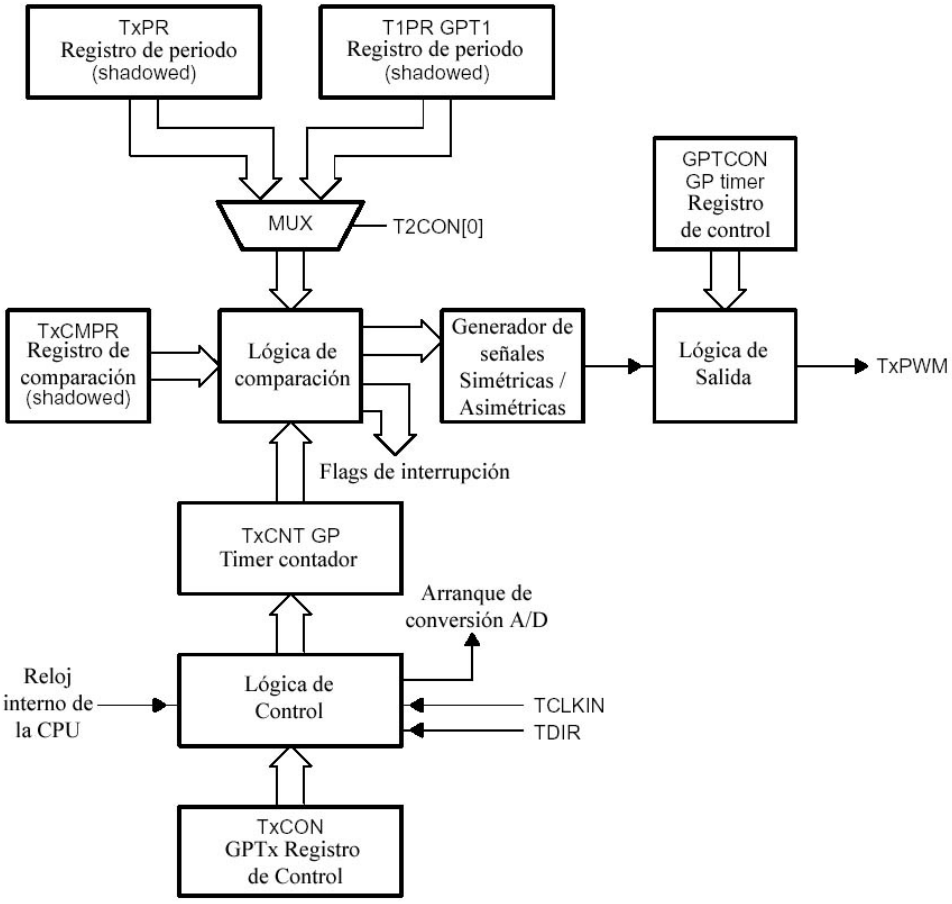


Figura 4.24. Diagrama de bloques de los temporizadores.

La técnica de doble buffer para el valor nuevo de los registros de comparación y periodo permiten al programa actualizar dichos valores en cualquier momento, con lo que se puede cambiar el periodo (frecuencia) y el ancho de pulso (duty cycle) de una señal PWM para el periodo siguiente.

De la figura 4.23, destacan las siguientes entradas a al temporizadores:

- Reloj interno de la CPU.
- Reloj externo a través del pin TCLKIN, con frecuencia máxima de ¼ de la frecuencia del reloj de la CPU. La selección del reloj se hace en el registro de control *TxCON*.
- Pin de entrada de dirección de la cuenta del temporizador TDIR, que indica si la cuenta es ascendente o descendente cuando el temporizador está configurado como contador ascentedente/descendente direccional (modo up/down direccional). La selección del modo se realiza a través del registro de control *TxCON*.
- Señal RESET del sistema

Cuando el temporizador se emplea junto con el circuito QEP, éste último genera tanto el reloj como la dirección de conteo.

Por otro lado, las salidas de los temporizadores son las siguientes:

- Pines de salida de comparación de cada temporizador TxCMP/TxPWM.
- Señal de inicio de la conversión A/D, la cual puede activarse por u rebosamiento del contador, por ocurrir una comparación o por llegar al periodo final de conteo. De esta forma se sincroniza el convertidor ADC con los temporizadores sin intervención de la CPU.
- Señales de rebasamiento (underflow y overfow), señal de comparación alcanzada y señal de periodo.
- Bits de indicación de la dirección de la cuenta.

Hay que destacar que cuando se produce un reset del sistema todos los bits de los registros asociados a los temporizadores y unidades de comparación se resetean a '0', al igual que los indicadores y máscaras de interrupción. Al mismo tiempo, se establecen las salidas de los temporizadores y unidades de comparación en un estado de alta impedancia.

4.6.1.1.Registro TxPR:

Éste registro contiene el valor del periodo para el temporizador correspondiente: *T1PR* para el Timer 1 y *T2PR* para el Timer 2. Están mapeados en la memoria de datos en las direcciones 7403h y 7407h respectivamente.

La operación de un temporizador se detiene y mantiene el valor actual, se resetea a '0' o comienza a contar hacia atrás cuando ocurre una coincidencia entre el registro del periodo y el registro contador, dependiendo del modo de conteo configurado del temporizador. Éste registro, al igual que el registro *TxCMPR* tienen un doble buffer, lo que permite cambiarles el valor en cualquier instante de tiempo.

El valor que se almacena en éste registro es el número de cuentas que debe realizar el contador hasta llegar a dicho valor. Como cada cuenta tarda un determinado tiempo en realizarse según el reloj del procesador, éste número de cuentas equivale a un periodo de tiempo para una señal periódica. Para generar una señal con un determinado periodo o frecuencia, el número de cuentas se calcula de la siguiente manera:

$$TxPR = \frac{T_{PWM}}{T_{clock} \cdot TPS} - 1$$

donde T_{PWM} es el periodo de la onda que queremos generar, T_{clock} es el periodo de la señal de reloj empleada (50ns para el TMS320F243) y TPS el preescalado del reloj que se ha configurado en el registro *TxCON*. Por ejemplo, para generar una onda cuadrada de frecuencia 10 kHz (T = 0.0001 s) con el TMS320F243 y un preescalado de 2, se obtendría: *TxPR* = (0.0001/50 exp(-9)*2)-1 = 999 cuentas. No hay gran diferencia entre 999 y 1000. En ambos casos un osciloscopio mide una frecuencia de 10 kHz.

Los bits que constituyen éste registro se detallan a continuación:

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Figura 4.25. Registros de periodo T1PR (7403h) y T2PR (7407h).

Nota: R = acceso a lectura; W = acceso a escritura; -0: valor después de reset.

- ***Bits 15-0: D15-D0.*** Valor almacenado indicando el número de cuentas a realizar por el temporizador..

4.6.1.2.Registro TxCMPR:

Se trata de los registros *T1CMPR* y *T2CMPR* asociados a los temporizadores 1 y 2 respectivamente. En ellos se almacena el valor que será comparado con el contenido por el registro contador correspondiente (*T1CNT* o *T2CNT*). Se produce una comparación cuando ambos valores coinciden (*TxCMPR* = *TxCNT*). Éste evento produce la transición en las señales PWM de salida del comparador, la activación del bit de interrupción y el inicio de la conversión A/D si está habilitada. La operación de comparación puede ser habilitada o deshabilitada configurando el bit 1 del registro de control *TxCON*.

Al igual que en el registro *TxPR*, aquí se almacena un valor que representa un número de cuentas. Como al darse una comparación se produce una transición en la señal y al llegar al periodo se vuelve al estado inicial, el valor que se almacene representará el ancho del pulso de la onda cuadrada generada (duty cylce), como se muestra en la figura 4.25. De esta forma, si queremos generar una onda de 10 kHz con duty cyce al 50%, se almacenaría el valor: *TxCMPR* = *TxPR*/2 = 1000/2 = 500 cuentas.

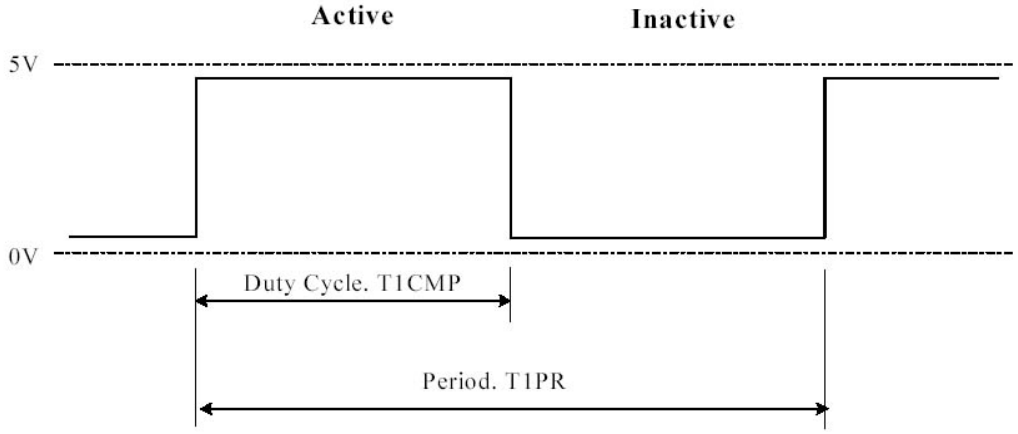


Figura 4.26. Duty cycle de onda cuadrada.

Los registros *T1CMPR* y *T2CMPR* se encuentran mapeados en la memoria de datos del procesador, en las direcciones 7402h y 7406h respectivamente. La estructura de éste registro es la siguiente:

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Figura 4.27. Registro comparador T1CMPR (7402h) y T2CMPR (7406h).

Nota: R = acceso a lectura; W = acceso a escritura; -0: valor después de reset.

- ***Bits 15-0: D15-D0.*** Valor almacenado para su comparación con el contador.

4.6.1.3.Registro TxCNT:

Los temporizadores disponen de dos registros contadores de 16 bits, *T1CNT* y *T2CNT*. Éstos registros contienen el valor actual del contador y lo incrementan o decrementan dependiendo de la dirección de conteo. Ambos registros se encuentran mapeados en el mapa de memoria de datos en las direcciones 7401h y 7405h.

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0

Figura 4.28. Registros contadores T1CNT (7401h) y T2CNT (7405h).

Nota: R = acceso a lectura; W = acceso a escritura; -0: valor después de reset.

- ***Bits 15-0: D15-D0.*** Valor actual del contador.

4.6.1.4.Registro TxCON:

Existen dos registros de control, uno para cada temporizador, que son configurables independientemente, y mapeados en la memoria de datos en las direcciones 7404h (*T1CON*) y 7408h (*T2CON*). Ambos se configuran de igual manera, y controlan el modo de funcionamiento de cada temporizador. Cada registro se divide en las siguientes secciones:

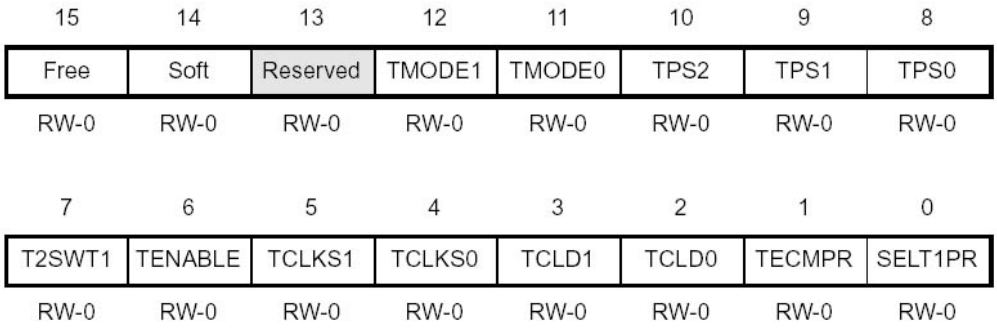


Figura 4.29. Registros de control T1CON (7404h) y T2CON (7408h).

Nota: R = acceso a lectura; W = acceso a escritura; -0: valor después de reset.

- ***Bits 15-14: Free, soft.*** Bits de control de la interrupción de la emulación. La suspensión de la emulación se produce cuando el reloj del dispositivo es detenido por el emulador, por ejemplo cuando el emulador encuentra un punto de ruptura o Brek Point. Las posibles combinaciones son:

  00 Se detiene inmediatamente si se suspende la emulación.
  01 Parar después de concluir el periodo actual del temporizador si se suspende la emulación.
  10 El temporizador no se ve afectado si se suspende la emulación.
  11 El temporizador no se ve afectado si se suspende la emulación.

- ***Bit 13: Reservado.*** Es leído como un cero. La escritura sobre él no tiene efecto.

- ***Bits 12-11: TMODE1-TMODE0.*** Bits de control del modo de coteo del temporizador. Las posibilidades son:

  00 Para el temporizador/mantiene el valor actual (Stop/Hold).
  01 Modo de cuenta continuo Up/Down.
  10 Modo de cuenta continuo Up.
  11 Modo de cuenta direccional Up/Down.

- ***Bits 10-8: TPS2-TPS0.*** Bits de control del preescalado del reloj de entrada al temporizador. De esta forma se obtiene una señal de reloj de menor frecuencia que la del reloj de la CPU. Las combinaciones posibles son:

  000 Frecuencia de reloj de la CPU//1
  001 Frecuencia de reloj de la CPU/2
  010 Frecuencia de reloj de la CPU/4
  011 Frecuencia de reloj de la CPU/8
  100 Frecuencia de reloj de la CPU/16
  101 Frecuencia de reloj de la CPU/32
  110 Frecuencia de reloj de la CPU/64
  111 Frecuencia de reloj de la CPU/128

- ***Bit 7: T2SWT1.*** Bit que decide la puesta en marcha del temporizador T2 con el bit de habilitación del temporizador T1. De esta forma pueden independizarse o coordinarse el funcionamiento de ambos temporizadores.

  0 usa su propio bit TENABLE
  1 Emplea el bit TENABLE del temporizador T1 (en *T1CON*) para permitir o inhabilitar la operación del temporizador T2 ignorando su propio bit de TENABLE.

- ***Bit 6: TENABLE.*** Determina la habilitación del temporizador.

  0 Deshabilita el temporizador y lo detiene.
  1 Habilita la acción del temporizador.

- ***Bits 5-4: TCLK1, TCLK0.*** Determina el origen de la señal de reloj del temporizador. Las posibilidades son:

  00 Interno.
  01 Externo.
  10 Reservado.
  11 Circuito QEP (en Timer 2) Reservado para el Timer 1.

- ***Bits 3-2: TCLD1, TCLD0.*** Indican la condición que provoca la recarga del registro de comparación del temporizador, *TxCMPR*. Las posibilidades son:

  00 Cuando la cuenta es igual a 0.
  01 Cuando la cuenta es igual a 0 o igual al valor almacenado en el registro de periodo.
  10 Inmediatamente.
  11 Reservado.

- ***Bit 1: TECMPR.*** Habilita la comparación del temporizador.

  0 Inhabilita la operación de comparación del temporizador.
  1 Habilita la operación de comparación del temporizador.

- ***Bit 0: SELT1PR.*** Registro de selección del registro de periodo para el Timer 2 (reservado para el Timer 1).

  0 Usa su propio registro de periodo (*T2PR*).
  1 Emplea el registro de periodo del temporizador 1 (*T1PR*).

4.6.1.5.Modos de conteo del temporizador:

Cada temporizador tiene cuatro posibles modos de operación dependiendo de la configuración de los bits 11 y 12 del registro de control TxCON, en los que se indica el tipo de conteo a realizar por el temporizador. Los modos son los siguiente:

- Stop/Hold.
- Modo de cuenta continuo ascendente (Up).
- Modo de cuenta direccional ascendente/descendente (Up/Down).
- Modo de cuenta continuo ascendente/descendente (Up/Down).

I. Stop/Hold:

En este modo de operación se detiene el temporizador y mantiene el valor actual. El contador, la salida de comparación y el preescalado no sufren ninguna variación.

II. Modo de cuenta continua ascendente (Up):

Configurando este modo de operación, el temporizador contará de forma ascendente hasta alcanzar el valor indicado en el registro de periodo, utilizando el reloj de entrada preescalado según la configuración establecida en el registro de control *TxCON*. Cuando el temporizador alcanza el valor del periodo, en el siguiente flanco ascendente de la señal de reloj, comienza de nuevo la cuenta ascendente desde cero (figura 4.29). Un ciclo de reloj después de llegar al periodo, se establece el indicador de interrupción por periodo a '1'. Si la interrupción correspondiente esta habilitada, se producirá una petición de interrupción por parte del periférico. Si está configurado para ello en el registro *GPTCON*, en el momento de poner a '1' el indicador de interrupción se envía una señal de inicio de conversión al ADC.

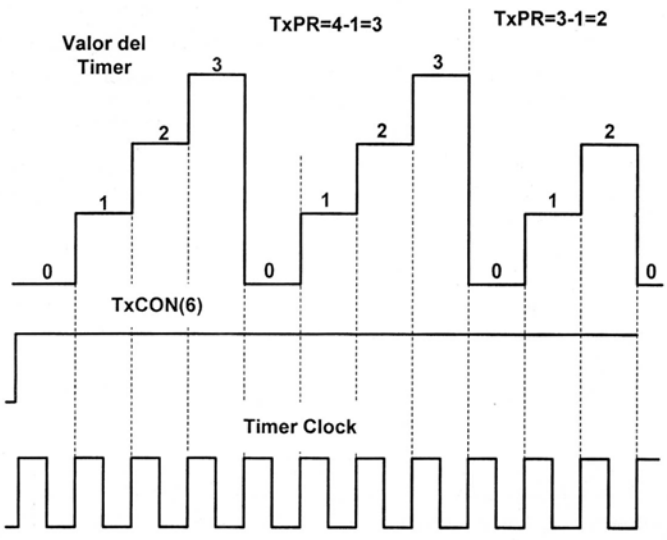


Figura 4.30. Modo de cuenta continua ascendente (Up).

También se puede producir una petición de interrupción por rebasamiento (underflow), es decir, cuando el contador llega al valor 0000h. Éste caso sigue el mismo proceso indicado anteriormente. Puede iniciar la conversión del ADC si está configurado para ello. Ocurre exactamente lo mismo cuando el contador sufre un overflow, es decir, llega al valor FFFFh. En todos los casos el indicador de interrupción se establece a '1' un ciclo de reloj después del evento, y se producirá la petición de interrupción si están permitidas. Con ello, cada vez que el contador llegue al periodo, se resete a cero o sufra desbordamiento, el bit indicador de interrupción correspondiente se establecerá a '1'.

El valor inicial del contador puede estar entre 0000h y FFFFh. Cuando es un valor intermedio cero y al periodo el temporizador cuenta hasta llegar al periodo para luego ponerse a cero y comenzar de nuevo. Si el valor inicial es mayor que el periodo, cuenta hasta llegar a FFFFh y se pone a cero para seguir contando.

El modo de conteo continuo ascendente es muy empleado en la generación de señales PWM asimétricas y para establecer los periodos de muestreo en sistemas de control de motores y movimiento. En la generación de señales PWM asimétrica (figura 4.30), los pines de salida *TxCMP/TxPWM* se mantienen inactivos antes de iniciar la cuenta, y permanecen en ese estado hasta que se produzca una igualdad entre el registro de comparación y el contador. En ese momento las salidas pasan a estado activo y permanecen así hasta llegar al final del periodo, momento en el que vuelven al estado inactivo, siempre y cuando el nuevo valor de comparación para el siguiente periodo no sea cero, ya que entonces la señal se mantendrá en el estado activo. Con estas características se pueden obtener señales con un duty cyce entre 0% y 100%. Si en algún caso el valor de comparación es mayor que el periodo, la salida será inactiva durante todo el periodo. Destacar, que tanto el estado activo como el inactivo pueden ser '0' o '1', según estén configurados en el registro *GPTCON*.

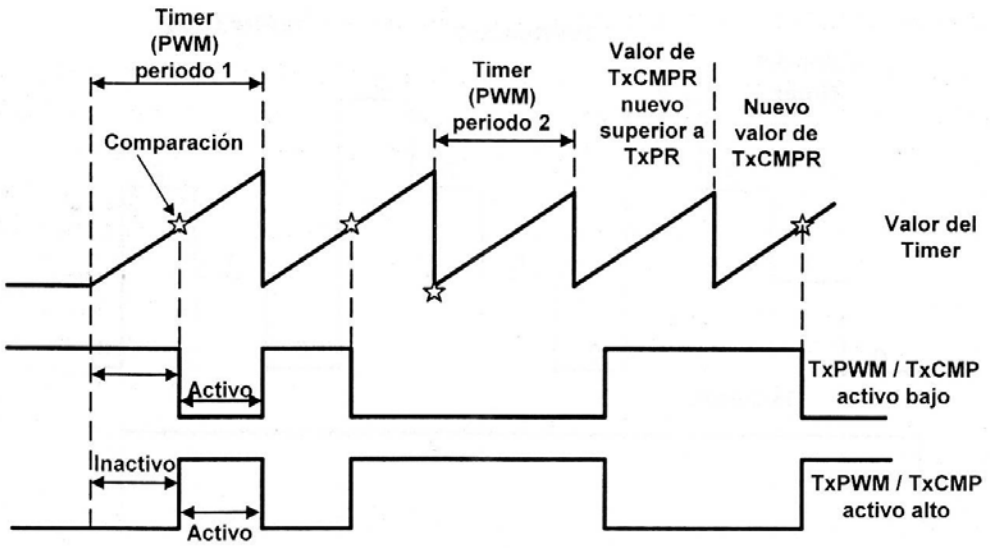


Figura 4.31. Generación de señales PWM asimétricas.

Las señales PWM asimétricas, se caracterizan porque al cambiar el valor del registro comparador sólo se ve afectado uno de los lados de la onda cuadrada. Por ejemplo, si una está configurada para ser activa a nivel bajo, para ampliar su duty cycle de 0% a 50% se cambiaría el valor del comparador con el valor correspondiente y, si mirásemos en un osciloscopio, el frente de subida de la señal se retrasaría, mientras el de bajada permanece en su lugar.

Como se comentó anteriormente, en la generación de señales PWM asimétricas, el valor a cargar en el registro de periodo se corresponde con el calculado mediante la expresión:

$$TxPR = \frac{T_{PWM}}{T_{clock} \cdot TPS} - 1$$

donde T_{PWM} es el periodo de la onda que queremos generar, T_{clock} es el periodo de la señal de reloj empleada (50ns para el TMS320F243) y TPS el preescalado del reloj que se ha configurado en el registro *TxCON*. También se comentó que el valor del registro comparador *TxCMPR* se correspondía con el ancho del pulso. Con estos dos valores se puede calcular los tiempos que la onda está en estado activo e inactivo:

$$t_{inactivo} = TxCMPR \cdot T_{clock} \cdot TPS$$
$$t_{activo} = \left[TxPR - TxCMPR + 1\right] \cdot T_{clock} \cdot TPS$$

III. Modo de cuenta direccional ascendente/descendente (Up/Down):

Otro modo empleado por los temporizadores es el de cuenta direccional ascendente/descedente, en el cual el temporizador cuenta incrementando o decrementando según el valor del pin de entrada TDIR. El funcionamiento en este caso es muy parecido al anterior. Si el pin de dirección se mantiene a nivel alto (TDIR = '1'), el modo de funcionamiento es exactamente el mismo que en modo de conteo continuo ascendente, es decir, cuando llega al periodo o a FFFFh el temporizador se resetea a cero para comenzar a contar de forma ascendente. Si por el contrario, el pin de dirección se encuentra a nivel bajo (TDIR = '0'), el contador cuenta decrementando hasta cero, momento en el que se recarga con el valor del periodo iniciando nuevamente la cuenta de forma descendente (figura 4.31).

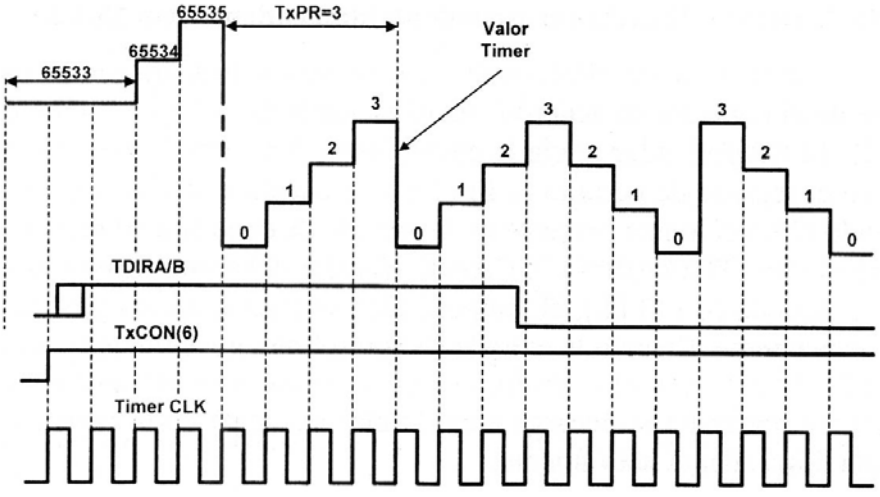


Figura 4.32. Cuenta direccional ascendente/descendente.

El tiempo que pasa desde que el pin TDIR cambia su valor hasta que cambia la dirección de la cuenta es de un ciclo de reloj después de la cuenta actual. La dirección de la cuenta es indicada por el temporizador en un bit del registro *GPTCON*: si es '1' es cuenta ascendente y '0' si es descendente.

El modo de cuenta ascendente/descendente direccional del temporizador 2 puede emplearse con los circuitos del codificador de pulsos de cuadratura QEP. Estos circuitos proporcionan el reloj de conteo y la dirección del temporizador 2 en este caso. Éste modo de trabajo puede aplicarse para temporizar eventos externos en control de motores y movimiento.

IV. Modo de cuenta continuo ascendente/descendente (Up/Down):

Éste modo de operación es exactamente el mismo que el del caso anterior, con la excepción de que en este caso el pin TDIR no tiene función alguna. Así, el contador realiza la cuenta hasta llegar al periodo de forma ascendente, después continua la cuenta de forma descendente hasta llegar a cero para volver a empezar de nuevo (figura 4.32). Si el valor del periodo es mayor que FFFFh, el contador cuenta hasta llegar a dicho valor, y después se resetea a cero continuando su operación como si el valor inicial fuese cero. Al igual que el caso anterior, la dirección de la cuenta es establecido en un bit del registro *GPTCON*.

El valor del periodo de esta señal es igual al doble del contenido del registro del registro de periodo (2**TxPR*), teniendo en cuenta el preescalado del reloj. Al igual que todos los casos anteriores, los temporizadores pueden emplear el reloj interno o uno externo procedente del pin TCLKIN.

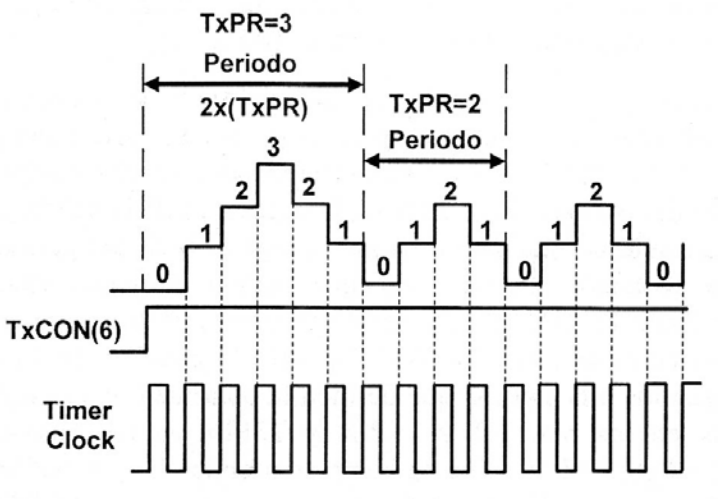


Figura 4.33. Modo de cuenta continuo ascendente/descendente.

En la generación de señales PWM simétricas se suele emplear este método. Éste tipo de señales se aplican en electrónica de potencia y el control de motores DC y AC.

Al emplear este modo en la generación de señales PWM simétricas, las salidas del comparador *TxPWM/TxCMP* permanecen inactivas hasta que no se de una primera comparación. En dicho momento se cambia al estado activo y se permanece en ese estado hasta llegar a la segunda comparación, cuando se vuelve al estado inactivo hasta finalizar el periodo (figura 4.33). En caso de que no hubiese segunda comparación las salidas se resetean al estado inactivo al final del periodo. Al igual que en los casos anteriores, el valor del registro de comparación puede variarse en cualquier momento, con lo que la primera comparación puede no coincidir con el valor de la segunda comparación.

Si el valor del registro de comparación fuese 0000h al principio de un periodo, la señal de salida se establecería a estado activo al principio del periodo hasta llegar la segunda comparación.

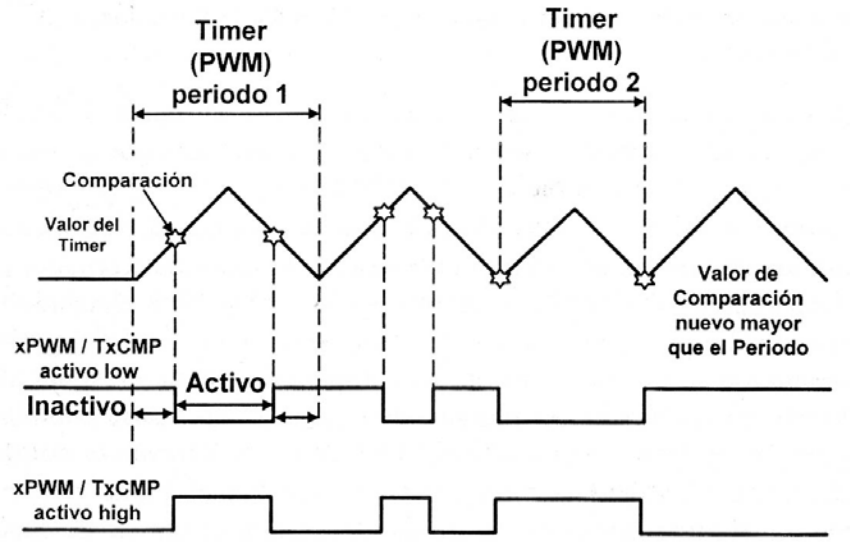


Figura 4.34. Señal PWM simétrica.

Como se comentó anteriormente, el periodo de la señal resultante es el doble que el valor cargado en el registro de periodo. Por ello, para calcular el valor que debe cargarse en dicho registros se emplea la siguiente expresión:

$$TxPR = \frac{T_{PWM}}{2 \cdot T_{clock} \cdot TPS} - 1$$

donde T_{PWM} es el periodo de la onda que queremos generar, T_{clock} es el periodo de la señal de reloj empleada (50ns para el TMS320F243) y TPS el preescalado del reloj que se ha configurado en el registro *TxCON*. En el modo de cuenta continua ascendente/descendente, cuando es aplicado a la generación de señales PWM simétricas, el valor del registro comparador puede ser diferente en el tramo ascendente y en el descendente. La variación de éste valor afecta a ambos lados de la señal cuadrada generada, ya que modifica tanto al flanco de subida como el de bajada. De ahí que se denominen simétricas. La duración de la salida en estado activo es:

$$t_{activo} = (TxPR - TxCMPR_{up} + TxR - TxCMPR_{down}) \cdot T_{clock} \cdot TPS$$

donde $TxCMPR_{up}$ es el valor del registro de comparación en la subida y $TxCMPR_{down}$ el de la bajada.

4.6.1.6.Registro de control GPTCON:

El Gestor de Eventos del procesador dispone de un registro de configuración global para los dos temporizadores de uso general que contiene. Éste registro se denomina *GPTCON* y está mapeado en la dirección de memoria de datos 7400h. Con él se configuran las distintas acciones a ejecutar en determinados eventos causados por los temporizadores.

Figura 4.35. Registro de control global de temporizadores GPTCON (7400h).

Nota: R = acceso de lectura; W = acceso de escritura; -x = valor después de un reset.

- ***Bit 15: Reservado.*** Es leído como un cero y no tiene efecto su escritura.

- ***Bit 14: T2STAT.*** Indica el estado del temporizador 2. Es un bit de solo lectura.

  0 Cuenta descendente.
  1 Cuenta ascendente.

- ***Bit 13: T1STAT.*** Indica el estado del temporizador 1. Es un bit de solo lectura.

  0 Cuenta descendente.
  1 Cuenta ascendente.

- ***Bits 12-11: Reservados.*** Son leídos como ceros y no tiene efecto su escritura.

- ***Bits 10-9: T2TOADC.*** Configura el inicio de conversión en el ADC dependiendo de lo que ocurra en el temporizador 2.

  00 Ningún evento inicia el ADC.
  01 La puesta a ‘1’ del flag de interrupción por underflow inicia el ADC.
  10 La puesta a ‘1’ del flag de interrupción por periodo inicia el ADC.
  11 La puesta a ‘1’ del flag de interrupción por comparación inicia el ADC.

- ***Bits 8-7: T1TOADC.*** Configura el inicio de conversión en el ADC dependiendo de lo que ocurra en el temporizador 1.

  00 Ningún evento inicia el ADC.
  01 La puesta a '1' del flag de interrupción por underflow inicia el ADC.
  10 La puesta a '1' del flag de interrupción por periodo inicia el ADC.
  11 La puesta a '1' del flag de interrupción por comparación inicia el ADC.

- ***Bit 6: TCOMPOE.*** Bit que permite la habilitación de la salida de comparación. Si el pin PDINT está activado, este bit se establece a cero inhabilitando todas las salidas de comparación de los temporizadores.

  0 Deshabilita todas las salidas de comparación.
  1 Habilita todas las salidas de comparación.

- ***Bits 5-4: Reservados.*** Son leídos como ceros y su escritura no tiene efecto.

- ***Bits 3-2: T2PIN.*** Bits que configuran la polaridad de la salida de comparación asociada al Timer 2.

  00 Salidas forzadas a nivel bajo.
  01 Activas a nivel bajo. Salida será 0 cuando *TxCNT>TxCMPR*.
  10 Activas a nivel alto. Salida será 1 cuando *TxCNT>TxCMPR*.
  11 Forzadas a nivel alto.

- ***Bits 1-0: T1PIN.*** Bits que configuran la polaridad de la salida de comparación asociada al Timer 1.

  02 Salidas forzadas a nivel bajo.
  03 Activas a nivel bajo. Salida será 0 cuando *TxCNT>TxCMPR*.
  10 Activas a nivel alto. Salida será 1 cuando *TxCNT>TxCMPR*.
  11 Forzadas a nivel alto.

Como se puede comprobar, mediante la configuración de éste registro, junto con el registros TxCON, se puede coordinar o no el funcionamiento de ambos temporizadores, es decir, que inicien la cuenta a la vez, empleen el mismo registro de periodo, etc, además de tener la posibilidad de iniciar una conversión en las entradas analógico-digitales. Esto último se emplea con bastante frecuencia, ya que con los temporizadores se puede programar un tiempo de muestreo para el convertido A/D, no teniendo así que estar el convertidor leyendo de forma continuada en las entradas analógico-digitales. También se pueden usar estos tiempos de muestreo para generar periódicamente interrupciones que se encarguen de comprobar o ejecutar cada cierto tiempo una rutina o estado del procesador. De esta forma se ocupan menos recursos del procesador, que pueden ser empleados en otras tareas que necesiten de más potencia de computación.

4.6.2. Unidades de comparación:

El Gestor de Eventos EV del procesador TMS320F243, dispone de tres unidades de comparación completas (unidades 1, 2 y 3).Cada una de estas unidades tiene asociadas dos señales de salida PWM, las cuales están complementadas, es decir, cuando una de las señales está a nivel alto, la otra se encuentra a nivel bajo y viceversa. La base de tiempos que emplean las unidades de comparación la da el temporizador de uso general T1, que puede operar en cualquiera de los modos de conteo analizados anteriormente. Para la utilización de las unidades de comparación se disponen de los siguiente bloques (figura 4.35):

- Tres registros de comparación de 16 bits con registros auxiliares de precarga asociados (shadowed), denominados *CMPR1, CMPR2* y *CMPR3*.
- Un registro de control de la comparación de 16 bits denominado *COMCON*.
- Un registro de control de acción de 16 bits denominado *ACTR*, y que dispone de registro auxiliar de precarga asociado (shadowed). Efectúa el control de las salidas PWM.
- Seis pines de salida PWM triestado (PWM1 a PWM6) procedentes de las unidades de comparación: 2 señales PWM de cada unidad de comparación.
- Lógica de control de salidas y de interrupciones..

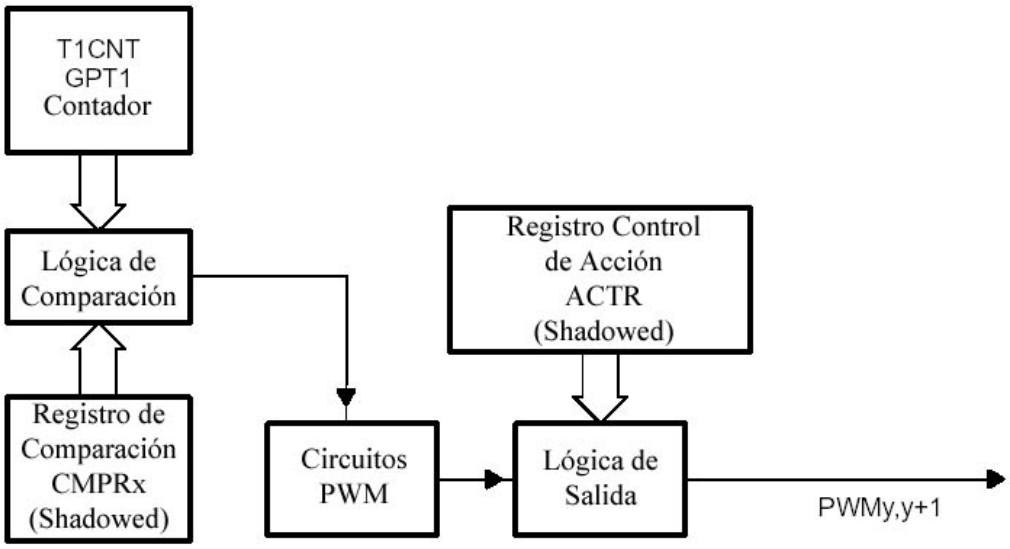


Figura 4.36. Diagrama de bloques de unidades de comparación.

Nota: x = 1, 2, 3; y = 1, 3, 5.

Las entradas a la unidad de comparación son:

- Las señales de control procedentes de los registros de control.
- El temporizador 1 así como sus señales de rebasamiento (underflow) y periodo.
- La señal de RESET del sistema.

El funcionamiento es muy parecido al de los temporizadores de uso general. El valor del temporizador T1 es comparado continuamente con el registro de comparación *CMPRx* correspondiente. Cuando se produce una coincidencia, se produce una transición en las dos salidas PWM asociadas a la unidad de comparación correspondiente un ciclo de reloj después de alcanzarse el valor, siendo esta transición acorde a lo configurado en el registro de acción *ACTR*. Los bits de éste registro pueden especificar individualmente el tipo de transición que debe tener cada una de las señales PWM cuando se da la comparación: activas o forzadas a nivel alto o bajo. Si la comparación está habilitada, el flag indicador de interrupción asociado a la unidad de comparación empleada (en registro *EVIFRx*) se establece a '1' un ciclo de reloj después de producirse la igualdad de valores en los registro contador y comparador. Entonces, se produce una petición de interrupción a la CPU si el bit de máscara correspondiente lo permite (en registro *EVAIMRx*). Todo este proceso se lleva a cabo de la misma forma que en los temporizadores de usos general, salvo que en este caso las salidas pueden ser modificadas por la lógica de salida, las unidades Dead-Band o la lógica de generación de señales PWM Space Vector.

El orden a seguir para configurar los registros cuando se desea utilizar las unidades de comparación es el siguiente:

- Cargar el registro *T1PR* de periodo con el valor deseado.
- Configurar el registro de acción *ACTR* con el valor adecuado.
- Configurar el registro de comparación *CMPRx* de la unidad deseada.
- Configurar el registro de control de comparación *COMCON*.
- Configurar el registro de control del temporizador 1 *T1CON* adecuadamente.

4.6.2.1. Registro COMCON:

La unidad de comparación dispone de un registro de control de la comparación, denominado *COMCON*, y mapeado en la dirección 7411h de la memoria de datos. Éste registro es de lectura y escritura y determina el modo de operación de la unidad de comparación, indicando:

- La habilitación o no de la operación de comparación.

- Si se habilitan o no las salidas de comparación.

- La condición de actualización de los registros de comparación con los valores existentes en sus registros auxiliares asociados (Shadowed).

- Si es habilitado el modo de espacio vectorial PWM o SVPWM.

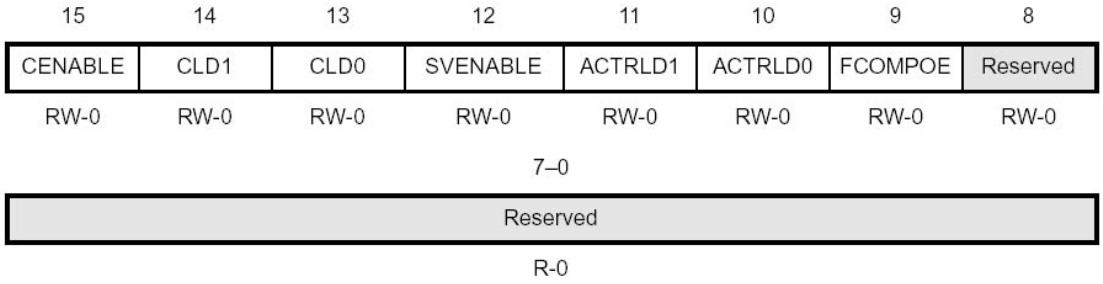


Figura 4.37. Registro de control de unidades de comparación COMCON (7411h).

Nota: R = acceso a lectura; W = acceso a escritura; -0 = valor después del reset.

- ***Bit 15: CENABLE.*** Bit de habilitación de la comparación.

  0 Deshabilita la operación de comparación y los registros auxiliares asociados a los registros *CMPRx* y *ACTR* son transparentes.
  1 Habilita la operación de comparación.

- ***Bits 14-13: CLD1, CLD0.*** Indican la condición de recarga del registro de comparación *CMPRx*.

  00 Cuando *T1CNT* = 0 (underflow).
  01 Cuando *T1CNT* = 0 (underflow) o cuando *T1CNT* = *T1PR* (se alcanza el periodo).
  10 Inmediatamente.
  11 Reservado. Resultado impredecible.

- ***Bit 12: SVENABLE.*** Bit que habilita el modo space vector PWM.

  0 Deshabilita el modo SVPWM.
  1 Habilita el modo SVPWM.

- ***Bits 11-10: ACTRLD1-ACTRLD0.*** Configuran la condición de recarga del registro de control de acción *ACTR*.

  00 Cuando *T1CNT* = 0 (underflow).
  01 Cuando *T1CNT* = 0 (underflow) o cuando *T1CNT* = *T1PR* (se alcanza el periodo).
  10 Inmediatamente.
  11 Reservado.

- ***Bit 9: FCOMPOE***. Habilita las salidas de comparación. Si la señal PDINT está activa este bit se pone a '0'.

  0 Los pines de salida PWM están inhabilitados y en alta impedancia.
  1 Los pines de salida PWM no están en alta impedancia, por tanto están habilitados.

- ***Bits 8-9: Reservados.*** Son leídos como ceros y la escritura no tiene efectos.

4.6.2.2. Registro ACTR:

Éste registro controla la acción que tiene lugar en cada una de las seis señales de salida PWM cuando ocurre una comparación, si están habilitadas en el registro *COMCON*. Está mapeado en la dirección 7413h de la memoria de datos. Éste registro tiene asociado un registro auxiliar de precarga (Shadowed). La condición en la que se recarga el valor del registro se configura en el registro de control *COMCON*. Éste registro dispone de cuatro bits que permiten la implementación de la modulación SVPWM.

15	14	13	12	11	10	9	8
SVRDIR	D2	D1	D0	CMP6ACT1	CMP6ACT0	CMP5ACT1	CMP5ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
CMP4ACT1	CMP4ACT0	CMP3ACT1	CMP3ACT0	CMP2ACT1	CMP2ACT0	CMP1ACT1	CMP1ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Figura 4.38. Registro de control de acción ACTR (7413h).

Nota: R = acceso a lectura; W = acceso a escritura; -0 = valor después del reset.

- ***Bit 15: SVDIR.*** Bit que indica la dirección de rotación del vector espacial en la modulación SVPWM. Solo se usa cuando se emplea esta técnica.

  0 Sentido de giro positivo (CCW o anti-horario).
  1 Sentido de giro negatico (CW o sentido horario).

- ***Bits 14-12: D2-D0.*** Bits correspondientes a los vectores espaciales básicos. Solo se emplea cuando se implementa esta técnica de modulación.

- ***Bits 11-10: CMP6ACT1-0.*** Configuran el pin de salida de comparación CMP6.

  00 Se fuerza la salida a nivel bajo.
  01 Salida activa a nivel bajo.
  10 Salida activa a nivel alto.
  11 Se fuerza la salida a nivel alto.

- ***Bits 9-8: CMP5ACT1-0.*** Configuran el pin de salida de comparación CMP5.

  00 Se fuerza la salida a nivel bajo.
  01 Salida activa a nivel bajo.
  10 Salida activa a nivel alto.
  11 Se fuerza la salida a nivel alto.

- ***Bits 7-6: CMP4ACT1-0.*** Configuran el pin de salida de comparación CMP4.

  00 Se fuerza la salida a nivel bajo.
  01 Salida activa a nivel bajo.
  10 Salida activa a nivel alto.
  11 Se fuerza la salida a nivel alto.

- ***Bits 5-4: CMP3ACT1-0.*** Configuran el pin de salida de comparación CMP3.

  00 Se fuerza la salida a nivel bajo.
  01 Salida activa a nivel bajo.
  10 Salida activa a nivel alto.
  11 Se fuerza la salida a nivel alto.

- ***Bits 3-2: CMP2ACT1-0.*** Configuran el pin de salida de comparación CMP2.

  00 Se fuerza la salida a nivel bajo.
  01 Salida activa a nivel bajo.
  10 Salida activa a nivel alto.
  11 Se fuerza la salida a nivel alto.

- ***Bits 1-0: CMP1ACT1-0.*** Configuran el pin de salida de comparación CMP1.

  00 Se fuerza la salida a nivel bajo.
  01 Salida activa a nivel bajo.
  10 Salida activa a nivel alto.
  11 Se fuerza la salida a nivel alto.

4.6.3. Circuitos para la generación de señales PWM:

El Gestor de eventos del TMS320F243, dispone de las siguientes prestaciones para la generación de ondas PWM:

- Dispone de cinco salidas PWM independientes, tres de ellas generadas por las unidades de comparación y otras dos generadas por los temporizadores de uso general. Por otro lado, las unidades de comparación pueden generar otras tres ondas dependientes de las tres anteriores, concretamente son las mismas señales pero complementadas.

- Capacidad para la programación de tiempos muertos para cada par de salidas PWM asociadas a las unidades de comparación (tiempo muerto mínimo de un ciclo de reloj).

- Valor mínimo del pulso PWM generado con incremento/decremento de un ciclo de reloj en los pulsos PWM generados.
- Resolución máxima de 16 bits en la generación de los pulsos PWM.
- Cambio de la frecuencia portadora PWM inmediato, al disponer de un registro de periodo auxiliar para la precarga.
- Cambio de los anchos de los pulsos PWM inmediatos, al disponer de registros comparadores auxiliares de precarga.
- Interrupción para la protección del circuito de excitación de la etapa de potencia (PDINT).
- Generación programable de señales PWM asimétricas, simétricas o con modulación de vectores espaciales (SVPWM).
- Mínima sobrecarga de la CPU debido a la autorecarga de los registro de comparación y periodo.

Como se acaba de detallar, las salidas de las unidades de comparación pueden ser modificadas por la lógica de salida, las unidades de tiempo muerto (Dead-Band) o por la lógica de modulación SVPWM. El diagrama de bloques de los circuitos PWM del Gestor de Eventos se muestra en la figura 4.38:

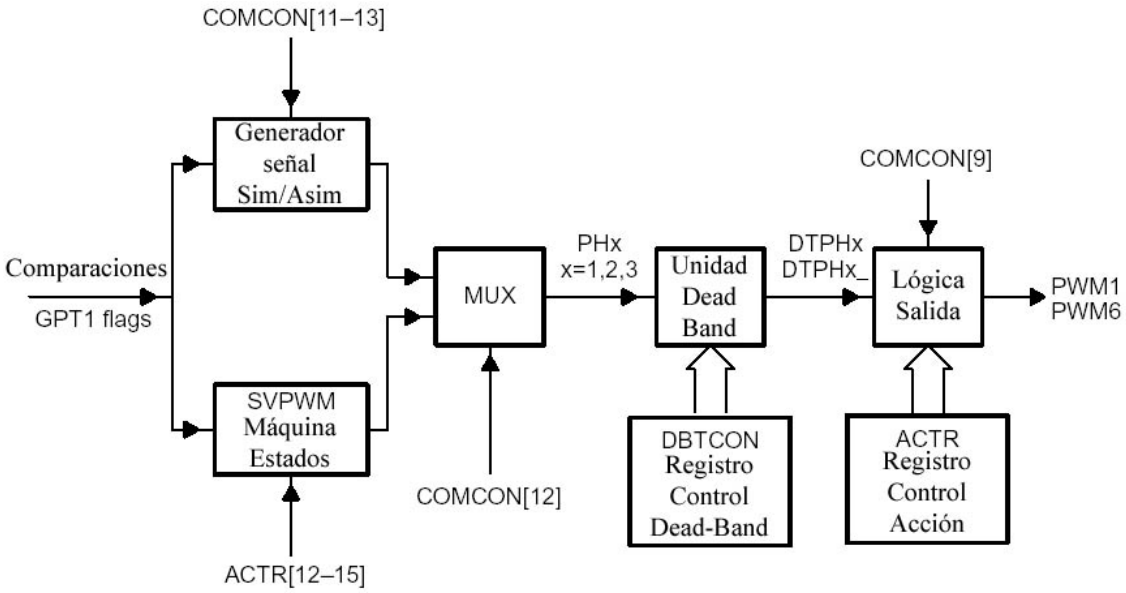


Figura 4.39. Diagrama de bloques de la unidad PWM.

Los circuitos PWM han sido diseñados para minimizar el uso de la CPU y reducir la intervención del usuario cuando se generan señales PWM. Ésta técnica es muy empleada en el control de motores y de movimiento. Los registros que controlan los circuitos de generación PWM son: *T1CON, COMCON, ACTR* y *DBTCON*.

El registro *DBTCON* es el responsable de programar los tiempos muertos en la generación de señales PWM. Está diseñada para prevenir el solapamiento de las señales PWM de los interruptores controlados de un mismo semipuente, evitando que los transistores superior e inferior del semipuente conduzcan a la vez, lo que provocaría un cortocircuito del bus de continua y podría destruir los dispositivos. Las características que presenta la unidad de tiempos muertos son:

- Dispone de un registro de control de 16 bits DBTCON.
- Existe la posibilidad de preescalado de la señal de reloj de la CPU.
- Dispone de tres contadores descendentes de 4 bits.
- Tiene una lógica de control asociada a la generación de tiempos muertos a la salida.

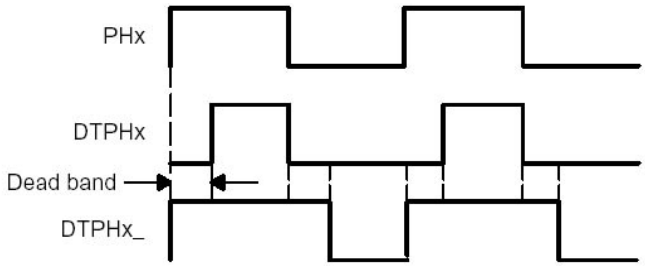


Figura 4.40. Señales de unidad de tiempos muertos.

Las entradas a la unidad Dead-Band se denominan PH1, PH2 y PH3, y proceden del generador de señales asimétricas/simétricas de las unidades de comparación 1, 2 y 3 respectivamente (figura 4.38). Para cada señal PHx , la unidad Dead-Band genera dos señales de salidas internas al DSP: DTPHx y DPTHx_. Si la unidad está deshabilitada ambas señales son iguales y complementadas, pero cuando está habilitada las transiciones en ambas señales están separadas por un intervalo de tiempo muerto (figura 4.39). La duración de este tiempo se configura en los bits 11 a 8 del registro *DBTCON* y por el preescalado del reloj, que se efectúa con los bits 4 a 2 del mismo registro. El intervalo de tiempo que se genera entre ambas señales es:

$$t_{dead-band} = (DBT3..0) \cdot (DBTPS2..0) \cdot T_{CPU\_clock}$$

Donde (DBT3..0) configuran el periodo del temporizador de la unidad Dead-Band (entre 0 y 16), (DBTPS2..0) proporciona el preescalado del temporizador (entre 1 y 32) y T_{CPU_clock} representa la duración de un ciclo de reloj de la CPU (50 ns en el TMS320F243).

A partir de las señales generadas por la unidad e tiempos muertos, los bits correspondientes del registro *COMCON* y la configuración del registro *ACTR*, la lógica de salida establece el estado y las transiciones de las salidas PWM. De esta forma pueden activarse a nivel alto o bajo, forzarse a nivel alto o bajo, presentar tiempos muertos, etc.

4.6.3.1. Registro DBTCON:

Configura el funcionamiento de la unidad de control de tiempos muertos en las señales PWM. Se encuentra mapeado en la dirección de memoria de datos 7415h.

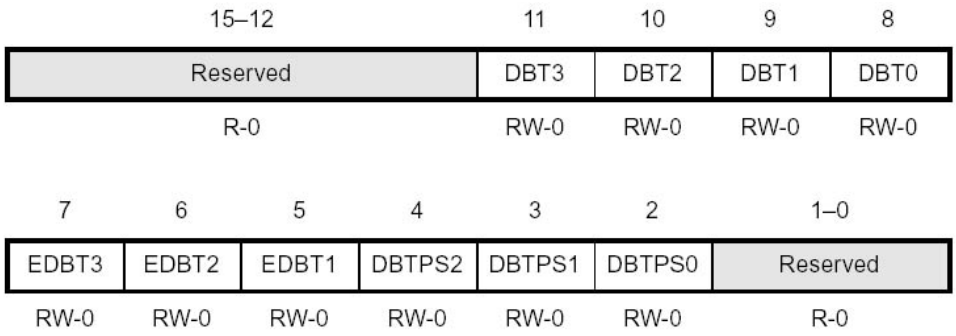


Figura 4.41. Registro de control de tiempos muertos DBTCON (7415h).

Nota: R = acceso a lectura; W = acceso a escritura; -0 = valor después del reset.

- ***Bits 15-12: Reservado.*** Son leídos como ceros y su escritura no tiene efectos.

- ***Bits 11-8: DBT3-DBT0.*** Bits que definen el periodo de los tres temporizador de 4 bits del Dead-Band.

- ***Bit 7: EDBT3.*** Bit de habilitación del temporizador 3 de la unidad Dead-Band, utilizado para los pines PWM5 y PWM6 de la unidad de comparación 3.

  0 Temporizador deshabilitado.
  1 Temporizador habilitado.

- ***Bit 6: EDBT2.*** Bit de habilitación del temporizador 2 de la unidad Dead-Band, utilizado para los pines PWM3 y PWM4 de la unidad de comparación 2.

  0 Temporizador deshabilitado.
  1 Temporizador habilitado.

- ***Bit 5: EDBT1.*** Bit de habilitación del temporizador 1 de la unidad Dead-Band, utilizado para los pines PWM1 y PWM2 de la unidad de comparación 1.

  0 Temporizador deshabilitado.
  1 Temporizador habilitado.

- ***Bits 4-2: DBTPS2 a DBTPS0.*** Bits de configuración del preescalado del temporizador.

  000 Frecuencia del reloj de la CPU/1
  001 Frecuencia del reloj de la CPU/2
  010 Frecuencia del reloj de la CPU/4
  011 Frecuencia del reloj de la CPU/8
  100 Frecuencia del reloj de la CPU/16
  101 Frecuencia del reloj de la CPU/32
  110 Frecuencia del reloj de la CPU/32
  111 Frecuencia del reloj de la CPU/32

- ***Bits 1-0: Reservados.*** Cuando se leen devuelven un cero y su escritura no tienen ningún efecto.

4.6.4. Unidades de Captura:

El gestor de eventos del procesador TMS320F243, dispone de tres unidades de captura (CAP1, CAP2 y CAP3) que permite detectar transiciones en el pin de entrada a cada unidad. Como base de tiempo, cada unidad puede elegir entre el temporizador 1 o 2, aunque CAP1 y CAP2 han de elegir el mismo. Su funcionamiento es capturando y almacenando el valor del temporizador utilizado en la pila FIFO de dos niveles cuando una transición especificada es detectada en el pin de entrada de captura (CAPx). Esto no afecta al funcionamiento de las unidades de comparación asociadas para PWM.

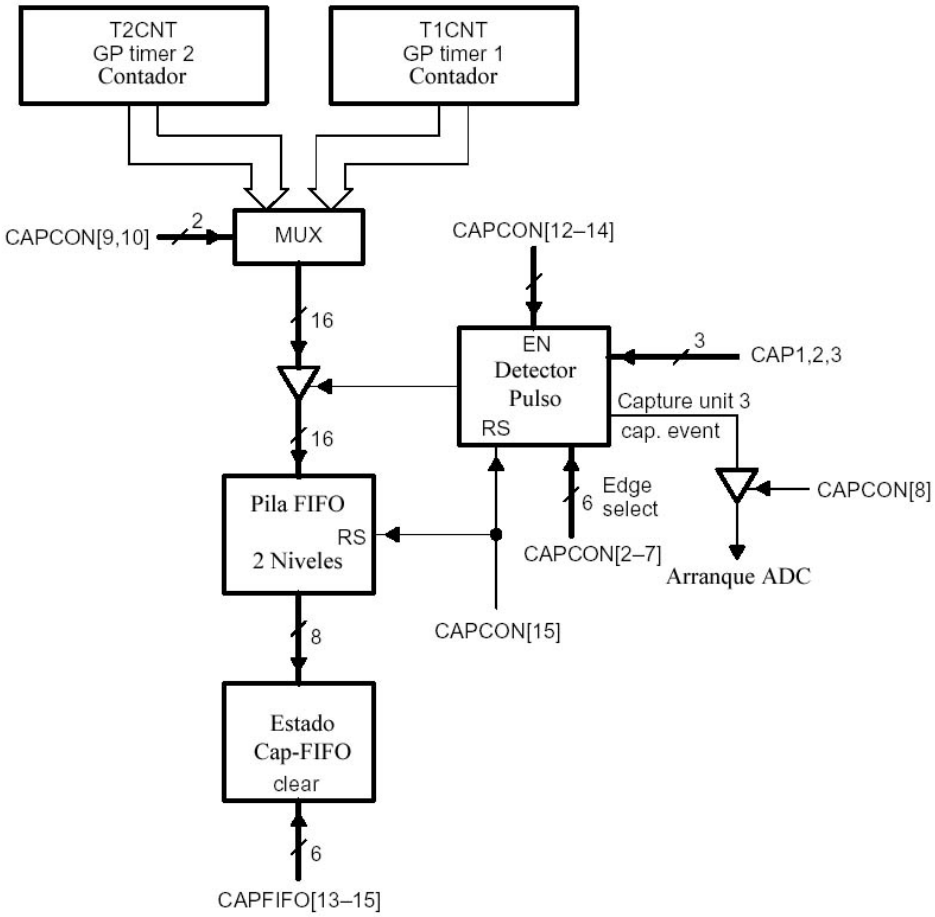


Figura 4.42. Diagrama de bloques de unidades de captura.

Las unidades de captura disponen de :

- Un registro de control de la captura de 16 bits denominado *CAPCON*.
- Un registro de estado de la FIFO de 16 bits llamado *CAPFIFO*.
- Selección entre los temporizadores 1 y 2 como base de tiempos.
- Una pila FIFO de 2 niveles de profundidad de 16 bits para cada captura.
- Tres pines de captura CAP1, CAP2 y CAP3, sincronizadas con el reloj de la CPU. Para capturar una transición esta debe mantenerse dos flancos de subida de reloj. CAP1 y CAP2 pueden usarse cono entradas QEP.
- Detección de transiciones definidas por el usuario: flancos de subida, flancos de bajada o ambos.
- Tres indicadores de interrupción enmascarabes, uno por unidad.

Cuando se produce un reset del sistema, todos los registros asociados a las unidades de captura se ponen a cero.

Para emplear las unidades de captura, éstas deben habilitarse. Si se produce la transición especificada en el pin correspondiente, el valor del temporizador elegido se guarda en la pila FIFO correspondiente. En ese instante se establece a '1' el indicador de interrupción correspondiente. Si está permitido, se genera la petición de interrupción. Además, los bits de estado de la FIFO se ajustan para reflejar el nuevo estado cada vez que se captura un valor nuevo. Desde que ocurre la transición hasta que el valor llega a la FIFO pasan dos ciclos de reloj.

Cada unidad de captura dispone de una pila FIFO de dos niveles de profundidad. La posición superior de la pila se denomina *CAPxFIFO* y la posición inferior *CAPxBOT*. La posición superior siempre contiene el valor más antiguo del contador capturado, y solo es de lectura. Cuando este dato es leído, el valor almacenado en el registro inferior de la FIFO pasa a ocupar el registro superior. Si la FIFO está vacía, el valor del contador capturado se almacena en el registro superior. El siguiente valor capturado se almacena en el registro inferior, y si el primer dato almacenado no es leído antes de producirse una tercera captura, éste se pierde al ser almacenado el segundo valor en el registro superior, ya que debe dejar el registro inferior para almacenar ese tercer valor capturado. Todos estos estados y perdidas de datos se expresan adecuadamente en los registro de estado de cada FIFO (*CAPFIFO*).

El orden de configuración de los distintos registros para el correcto funcionamiento de la unidad de captura es:

- Inicializar *CAPFIFOx* y limpiar los bits de estado correspondientes.
- Establecer el temporizador en uno de sus modos de funcionamiento.
- Establecer el registro de comparación o el registro de periodo del temporizador asociado.
- Configurar CAPCON apropiadamente.

4.6.4.1. Registro CAPCON:

La operación de las unidades de captura es configuran mediante un registro de 16 bits denominado *CAPCON*, mapeado en la memoria de datos en la dirección 7420h. Los registros de control de los temporizadores (*T1CON* y *T2CON*) también se usan para controlar el funcionamiento de las unidades de captura, ya que estos temporizadores se usan como base de tiempos. Éste registro también controla el circuito QEP.

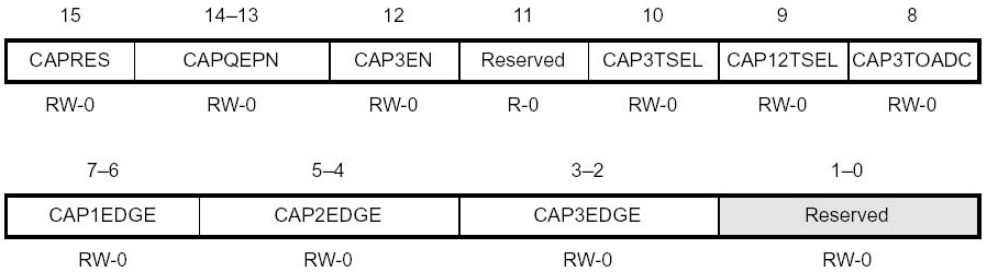


Figura 4.43. Registro de control CAPCON (7420h).

- ***Bit 15: CAPRES.*** Reset de la captura. Siempre es leído a cero.

  0 Pone a cero todos los registros de la unidad de captura y circuitos QEP.
  1 No tiene efecto.

- ***Bits 14-13: CAPQEPN:*** Bits de control de las unidades de captura 1 y 2, además de controlar el circuito QEP.

  00 Inhabilita las unidades de captura 1 y 2 y el circuito QEP. La pila FIFO mantiene sus contenidos.
  01 Habilita las unidades de captura 1 y 2 e inhabilita el circuito QEP.
  10 Reservado.
  11 Habilita el circuito QEP e inhabilita las unidades de captura 1 y 2.

- ***Bit 12: CAP3EN.*** Configura la unidad de captura 3.

  0 Inhabilita la unidad de captura 3 manteniendo los contenidos de su pila FIFO.
  1 Habilita la unidad de captura 3.

- ***Bit 11: Reservado***. Devuelve un cero al leerlo y su escritura no tiene efecto.

- ***Bit 10: CAP3TSEL.*** Selecciona el temporizador que será asociado a la unidad de captura 3.

  0 Selecciona el temporizador 2.
  1 Selecciona el temporizador 1.

- ***Bit 9: CAP12TSEL.*** Selecciona el temporizador que será asociado a las unidades de captura 1 y 2.

  0 Selecciona el temporizador 2.
  1 Selecciona el temporizador 1.

- ***Bit 8: CAP3TOADC.*** Configura la unidad de captura 3 para iniciar al ADC.

  0 No realiza acción.
  1 Inicia el ADC cuando el indicador CAP3INT se ponga a ‘1’.

- ***Bits 7-6: CAP1EDGE.*** Bits de control del flanco detectado para la unidad 1.

  00 No hay detección.
  01 Se detectan. Flancos de subida.
  10 Se detectan flancos de bajada.
  11 Se detectan flancos de subida y bajada.

- ***Bits 5-4: CAP2EDGE.*** Bits de control del flanco detectado para la unidad 2.

  00 No hay detección.
  01 Se detectan. Flancos de subida.
  10 Se detectan flancos de bajada.
  11 Se detectan flancos de subida y bajada.

- ***Bits 3-2: CAP3EDGE.*** Bits de control del flanco detectado para la unidad 3.

  00 No hay detección.
  01 Se detectan. Flancos de subida.
  10 Se detectan flancos de bajada.
  11 Se detectan flancos de subida y bajada.

- ***Bits 1-0: Reservados.*** Devuelve un cero al leerlo y su escritura no tiene efecto.

4.6.4.2. Registro CAPFIFO:

El registro *CAPFIFO* contiene los bits de estado de cada una de las tres pilas FIFO de las unidades de captura. Esta ubicada en la dirección 7422h de la memoria de datos. Si se escribe en un bit de estado a la vez que está siendo actualizado, el dato que se escribe tiene prioridad.

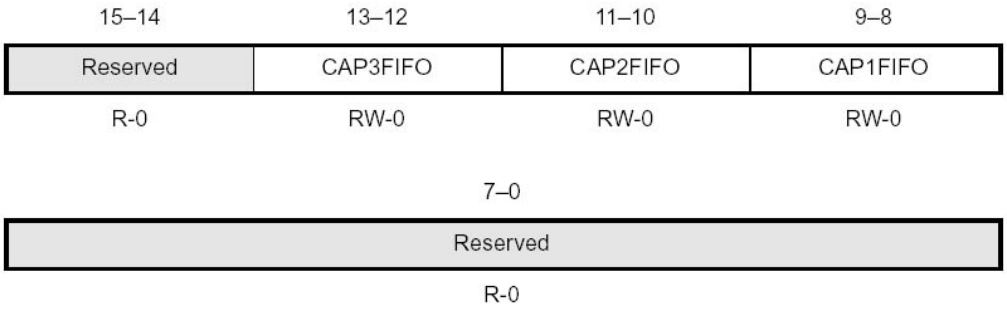


Figura 4.44. Registro de estado CAPFIFO (7422h).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***Bits 15-14: Reservados.*** Devuelve un cero al leerlo y su escritura no tiene efecto.

- ***Bits 13-12: CAP3FIFO.*** Bits de estado de *CAP3FIFO*.

  00 Vacía (no hay datos).
  01 Hay una entrada.
  10 Hay dos entradas.
  11 Hay dos entradas y otra captura más. La primera se ha perdido.

- ***Bits 11-10: CAP2FIFO.*** Bits de estado de *CAP2FIFO*.

  00 Vacía (no hay datos).
  01 Hay una entrada.
  10 Hay dos entradas.
  11 Hay dos entradas y otra captura más. La primera se ha perdido.

- ***Bits 9-8: CAP1FIFO.*** Bits de estado de *CAP1FIFO*.

  00 Vacía (no hay datos).
  01 Hay una entrada.
  10 Hay dos entradas.
  11 Hay dos entradas y otra captura más. La primera se ha perdido.

- ***Bits 7-0: Reservados.*** Su lectura devuelve un cero y su escritura no tiene efecto.

4.6.5. Circuito Quadrature Encoder Pulse (QEP):

El gestor de eventos contiene un circuito de pulsos de cuadratura o QEP, que cuando está habilitado, decodifica y cuenta los pulsos de entrada codificados en cuadratura existentes en los pines CAP1/QEP0 y CAP2/QEP1. Éste circuito puede usarse como elemento de conexión con un encoder óptico para obtener la posición y velocidad de una máquina rotatoria. Cuando se habilita el circuito QEP, la función de captura de los pines CAP1 y CAP2 es deshabilitada al ser pines compartidos. Para habilitar o no el circuito QEP, debe configurarse adecuadamente el registro *CAPCON*.

La base de tiempos para el circuito QEP es el temporizador 2, el cual debe estar configurado en modo direccional ascendente/descendente y con el circuito QEP como fuente de reloj.

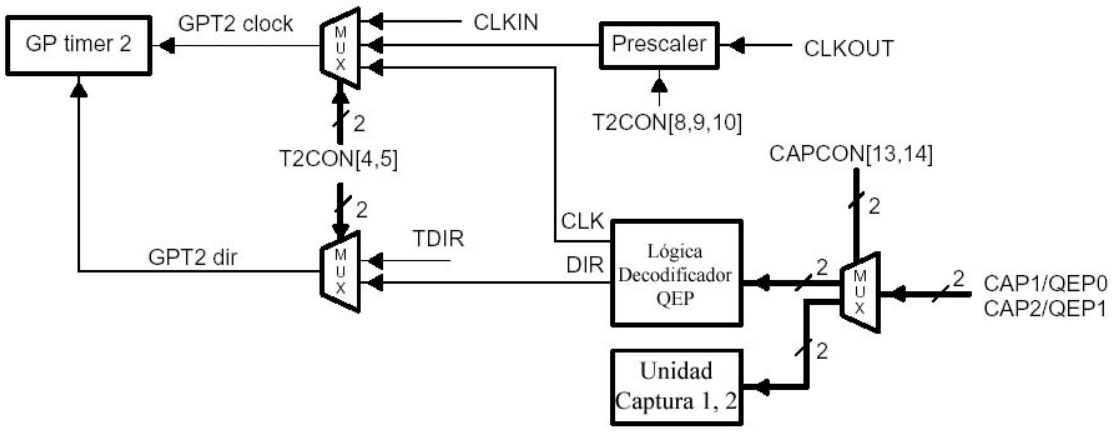


Figura 4.45. Diagrama de bloques de circuito QEP.

Los pulsos en cuadratura son dos secuencias de pulsos de frecuencia variable y un desfase fijo de 90 grados (figura 4.45). Cuando los genera un encoder óptico situado en el eje de un motor, se puede determinar la dirección de rotación detectando cual de las dos secuencias es la adelantada. También se pueden hallar la posición angular y la velocidad del motor contando los pulsos y determinando la frecuencia de estos. La lógica de detección del circuito QEP determina cual de las dos secuencias es la adelantada. Con ello, el gestor de eventos genera una señal de dirección como entrada al temporizador 2 (figura 4.44), el cual cuenta de forma ascendente si la entrada CAP1/QEP0 es la adelantada, y cuenta de forma descendente si la entrada CAP2/QEP1 es la secuencia adelantada.

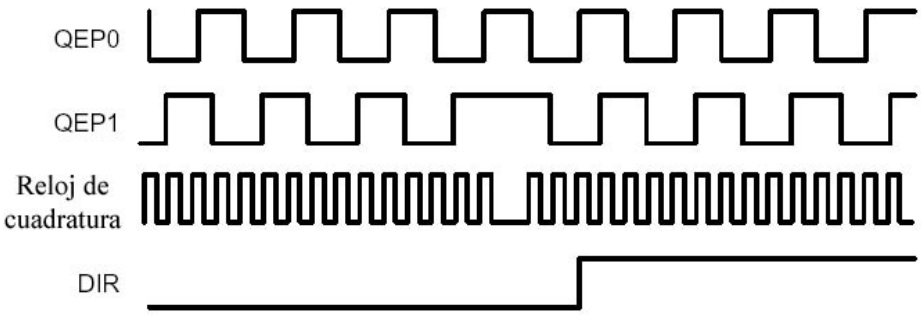


Figura 4.46. Pulsos de encoder de cuadratura.

En este tipo de detección, se cuentan los dos flancos de entrada procedentes de un encoder óptico. Por tanto, la frecuencia de reloj generada por la lógica QEP para el temporizador 2 es cuatro veces la de cada secuencia de entrada. Éste reloj de cuadratura se conecta a la entrada de reloj de los temporizadores.

Por otro lado, los indicadores de interrupción por periodo, underflow, overflow y comparación de los temporizadores, utilizando el reloj del circuito QEP, se activan en los eventos correspondientes. Éstos indicadores pueden generar petición de interrupción si están habilitadas.

Para comenzar el funcionamiento del circuito QEP deben seguirse estos pasos:

- Cargar el temporizador 2 con los valores deseados de periodo y los registros de comparación, si es necesario.
- Configurar *T2CON* en modo direccional ascendente/decendente empleando el circuito QEP como fuente de reloj y habilitar el timer escogido.
- Configurar el registro *CAPCON* para habilitar el circuito QEP.

4.6.6. Interrupciones en el Gestor de Eventos:

Las interrupciones del gestor de eventos del procesador TMS320F243, están agrupadas en tras grupos: A, B y C. Cada grupo tiene su registro de indicadores de interrupción (*EVIFRx*) y de habilitación de las mismas (*EVIMRx*). Un indicador de lso registros *EVIFRx* no genera interrupción, cuando el bit correspondiente del registro de máscaras *EVIMRx* está a '0'.

El procedimiento de atención de interrupciones del gestor de eventos es el siguiente:

1- Cuando ocurre una interrupción en el gestor de eventos, el indicador correspondiente a la interrupción se establece a '1' en el registro *EVIFRx* correspondiente.

2- Se genera una petición de interrupción por periférico al controlador PIE si el bit correspondiente del registro de máscaras *EVIMRx* está a '1' (habilitada).

3- Cuando la CPU reconoce la interrupción, el controlador PIE carga el vector de interrupción correspondiente de mayor prioridad pendiente en el registro PIVR.

4- Se atiende la rutina de interrupción pertinente. En ella, el bit de indicación de interrupción del periférico correspondiente, debe ser borrado escribiendo un '1' por software para permitir futuras interrupciones del mismo.

Los eventos que generaban interrupción en el gestor de eventos se resumen en: overflow, underflow, comparación y llegada al periodo. En la siguiente tabla se muestran las interrupciones posibles del gestor de eventos.

Grupo	Interrupción	Prioridad dentro del grupo	Vector (ID)	Descripción/fuente	INT
	PDPINT	1 (+)	0020h	Power Drive Protection Interrupt A	1
A	CMP1INT	2	0021h	Int. por comparación en unidad 1	2
	CMP2INT	3	0022h	Int. por comparación en unidad 2	
	CMP3INT	4	0023h	Int. por comparación en unidad 3	
	T1PINT	5	0027h	Int. por periodo en Timer 1	
	T1CINT	6	0028h	Int. por comparación en Timer 1	
	T1UFINT	7	0029h	Int. por underflow en Timer 1	
	T1OFINT	8 (-)	002Ah	Int. por overflow en Timer 1	
B	T2PINT	1 (+)	002Bh	Int. por periodo en Timer 2	3
	T2CINT	2	002Ch	Int. por comparación en Timer 2	
	T2UFINT	3	002Dh	Int. por underflow en Timer 2	
	T2OFINT	4 (-)	002Eh	Int. por overflow en Timer 2	
C	CAP1INT	1 (+)	0033h	Interrupción en unidad de captura 1	4
	CAP2INT	2	0034h	Interrupción en unidad de captura 2	
	CAP3INT	3	0035h	Interrupción en unidad de captura 3	

Tabla 4.6. Tabla de interrupciones del Gestor de Eventos.

4.6.6.1. Registros EVIFRA, EVIFRB y EVAFRC:

El gestor de eventos dispone de tres registros de indicadores de interrupción: *EVIFRA* (742Fh), *EVIFRB* (7430h) y *EVIFRC* (7431h), los cuales se encuentran mapeados en memoria. Cada uno de ellos se encargan de los distintos eventos posibles en cada uno de los grupos de interrupciones descritos en la tabla 4.6. Los tres tienen la misma estructura: Cada uno de sus bits se encargan de indicar una de las interrupciones indicadas en la tabla anterior. Si al leer un bit, éste vale '1', indica que ha habido una interrupción del tipo al que corresponda ese bit según la tabla 4.6. Si se escribe un '1' en un bit, se borra el indicador de interrupción de ese bit.

15–11	10	9	8
Reserved	T1OFINT	T1UFINT	T1CINT
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T1PINT	Reserved	CMP3INT	CMP2INT	CMP1INT	PDPINT
RW-0	R-0	RW-0	RW-0	RW-0	RW-0

Figura 4.47. Registro de indicadores de interrupción EVIFRA (742Fh).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***EVIFRA:*** Contiene los indicadores de interrupción del Timer1 y de los tres comparadores.

15–4	3	2	1	0
Reserved	T2OFINT	T2UFINT	T2CINT	T2PINT
R-0	RW-0	RW-0	RW-0	RW-0

Figura 4.48. Registro de indicadores de interrupción EVIFRB (7430h).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***EVIFRB:*** Contiene los indicadores de interrupción del Timer2.

15–3	2	1	0
Reserved	CAP3INT	CAP2INT	CAP1INT
R-0	RW-0	RW-0	RW-0

Figura 4.49. Registro de indicadores de interrupción EVIFRC (7431h).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***EVIFRC:*** Contiene los indicadores de interrupción de las tres unidades de captura.

4.6.6.2. Registros EVIMRA, EVIMRB y EVIMRC:

El gestor de eventos dispone de una tres registros de máscaras de las distintas interrupciones descritas en la tabla 4.6. Estos son: *EVIMRA* (742Ch), *EVIMRB* (742Dh) y *EVIMRC* (742Eh). En ellos se encuentran as máscaras de interrupción de los registros *EVIFRA*, *EVIFRB* y *EVIFRC*. Cada uno de sus bit se corresponden con una de las interrupciones de la tabla anterior. Cuando en un bit se tiene un '0', la interrupción correspondiente está deshabilitada. Si por el contrario tiene un '1', la interrupción se encuentra habilitada.

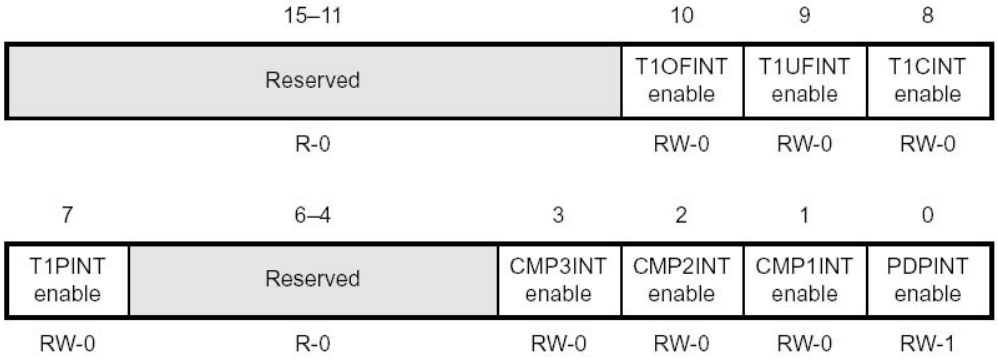


Figura 4.50. Registro de máscaras de interrupción EVIMRA (742Ch).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***EVIMRA:*** Contiene las máscaras de interrupción de Timer1 y de los tres comparadores.

Figura 4.51. Registro de máscaras de interrupción EVIMRB (742Dh).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***EVIMRB:*** Contiene las máscaras de interrupción de Timer2.

Figura 4.52. Registro de máscaras de interrupción EVIMRC (742Eh).

Nota: R = acceso a lectura; = acceso a escritura; -0 = valor después de reset.

- ***EVIMRC:*** Contiene las máscaras de interrupción de las tres unidades de captura..

4.6.7. Ejemplos de programación:

A continuación serán descritos una serie de aplicaciones para la mayor comprensión del empleo y configuración de los periféricos analizados anteriormente. En particular, ya ha se han empleado algunos de ellos en los ejemplos 3 y 4. En estos ejemplos se hacía uso de los temporizadores de uso general para generar interrupciones. Para ello, se configuraba el temporizador correspondiente para que realizara la cuenta de forma ascendente, de manera que cuando llegara a su periodo se generase una interrupción, la cual estaba habilitada por el bit correspondiente del registro de máscaras.

4.6.7.1. Ejemplo 3. Continuación:

En este ejemplo se trataba de mostrar en unos leds conectados al puerto D el estado de unos microinterruptores conectados al puerto B. En el proceso, se genera un tiempo de retraso entre cada acceso a los puertos. Éste tiempo de retraso y el acceso a los puertos se programó para que se llevara a cabo mediante una interrupción generada por el temporizador 1, de forma que la rutina de interrupción se encargara de ambos procesos, minimizando el uso de la CPU. Para llevar a cabo todo este proceso, los registros configurados fueron:

- ***GPTCON:*** Se configura el registro de control global de los temporizadores para indicar la polarización del Timer 1 como forzada a nivel bajo.

- ***T1CON:*** Se establece el modo de conteo en continuo ascendente, con fuente de reloj interno y prescaler CPUCLK/128. Se habilita también la operación de comparación. En un primer momento, mientras dure la configuración e iniciación del DSP, el temporizador no debe habilitarse mediante su bit T1CON_TENABLE (bit 6). Esto se hace justo al final, antes de empezar la rutina principal del programa, evitando así generar procesos antes de terminar la iniciación.

- ***T1PR:*** Se configura el registro de periodo del Timer 1 para generar una señal de 0.2 s de periodo. Para ello se emplea la expresión:

$$TxPR = \frac{T_{PWM}}{T_{clock} \cdot TPS} - 1$$

  Teniendo en cuente que el reloj de la CPU es de 50 ns y se emplea un prescaler de 128, para generar una señal de 0.2 s es necesario realizar: 0.2 s/(50 ns * 128) = 31250 cuentas en el temporizador.

- ***T1CNT:*** Se almacena el valor inicial para el contador.

- ***EVIMRA:*** Se procede a habilitar la interrupción por periodo en el Timer 1 (T1PINT, bit 7). Los demás registros *EVIMRx* se ponen a cero al no emplearse más interrupciones del gestor de eventos.

- ***EVIFRA:*** Por último se borran todas las posibles interrupciones en espera en todos los registros *EVIFRx*, escribiendo en todos sus bits un ‘1’.

Configurando así el funcionamiento del temporizador 1, cada 0.2 segundos se produce una interrupción por periodo. Lo que se realice después de esta interrupción sólo depende de lo que haya programado el usuarios. En este caso, en la rutina de interrupción T1PER_ISR definida, se decide esperar a que ocurran 5 interrupciones por periodo para generar un tiempo de retraso de 1 segundo antes de acceder a los puertos B y D. Lo último a realizar obligatoriamente en una rutina de interrupción después de ejecutar el proceso de la misma, es borrar el indicador flag de dicha interrupción en el registro de indicadores de interrupciones del periférico empleado, que en éste caso es el bit 7 del registro *EVIFRA*.

4.6.7.2. Ejemplo 4. Continuación:

Éste ejemplo mostraba un efecto de luz en movimiento en los leds conectados al puerto D. Para controlar la velocidad del movimiento era necesario generar un tiempos de retraso de 0.1 s entre posición y posición del led encendido. Ello se realizaba a través de una interrupción por periodo del temporizador 2, y cuya rutina era la encargada, además, de mostrar la secuencia de leds en el puerto D. También se emplea la señal del Timer 2 para generar una onda cuadrada en el pin de salida T2CMP/T2PWM del 50% de duty cycle. El tratamiento de la interrupción y de la rutina se realiza de manera semejante a lo descrito en el ejemplo anterior. De esta forma, los registro debían configurarse de la siguiente manera:

- ***GPTCON:*** Se configura el registro de control global de los temporizadores para indicar la polarización del Timer 2 como activa a nivel bajo.

- ***T2CON:*** Se establece el modo de conteo en continuo ascendente, con fuente de reloj interno y prescaler CPUCLK/64. Se habilita también la operación de comparación. En un primer momento, mientras dure la configuración e iniciación del DSP, el temporizador no debe habilitarse mediante su bit T2CON_TENABLE (bit 6). Esto se hace justo al final, antes de empezar la rutina principal del programa, evitando así generar procesos antes de terminar la iniciación. También se configura para emplear su propio in de habilitación y periodo.

- ***T2PR:*** Se configura el registro de periodo del Timer 2 para generar una señal de 0.1 s de periodo. Para ello se emplea la expresión:

$$TxPR = \frac{T_{PWM}}{T_{clock} \cdot TPS} - 1$$

  Teniendo en cuente que el reloj de la CPU es de 50 ns y se emplea un prescaler de 64, para generar una señal de 0.2 s es necesario realizar: 0.1 s/(50 ns * 64) = 31250 cuentas en el temporizador.

- ***T2CNT:*** Se almacena el valor inicial para el contador.

- ***T2CMPR:*** Se configura el registro comparador con la mitad del periodo para generar una onda de 50% de duty cycle.

- ***EVIMRB:*** Se procede a habilitar la interrupción por periodo en el Timer 2 (T2PINT, bit 0). Los demás registros *EVIMRx* se ponen a cero al no emplearse más interrupciones del gestor de eventos.

- ***EVIFRB:*** Por último se borran todas las posibles interrupciones en espera en todos los registros *EVIFRx*, escribiendo en todos sus bits un '1'.

4.6.7.3. Ejemplo 5:

Éste ejemplo produce en la salida T1PW/T1CMPR una señal PWM asimétrica de ancho variable y de frecuencia 80 kHz. Puede ser medida por un osciloscopio conectándolo al pin 4 del puerto B de la placa de pruebas desarrollada. Está diseñado para variar el duty cycle al vuelo con valores de comparación almacenados en tablas. Es una muestra de las facilidades que presenta el procesador TMS320F243 para generar ondas PWM, ya que permite modificar al vuelo tanto la frecuencia como el ancho de la onda. Para ello se emplea una rutina de interrupción por comparación del temporizador 1, es decir, que se ejecuta cada vez que el registro contador y comparador contienen el mismo valor. En dicha rutina se cambia el valor del registro comparador por el siguiente valor contenido en la tabla para modificar el ancho del pulso a partir del periodo siguiente. Para ello debe establecerse la siguiente secuencia de configuración de registros:

- ***OCRA:*** Se configura el registro de control de las señales de entrada/salida para seleccionar la función T1PWM del pin correspondiente.

- ***GPTCON:*** Se configura el registro de control global de los temporizadores para indicar la polarización del Timer 1 como activa a nivel bajo y habilitar las salidas de comparación.

- ***T1CON:*** Se establece el modo de conteo en continuo ascendente, con fuente de reloj interno y prescaler CPUCLK/1. Se habilita también la operación de comparación. En un primer momento, mientras dure la configuración e iniciación del DSP, el temporizador no debe habilitarse mediante su bit T1CON_TENABLE (bit 6). Esto se hace justo al final, antes de empezar la rutina principal del programa, evitando así generar procesos antes de terminar la iniciación. También se programa la condición de recarga de éste registro para cuando el contador alcance el periodo o sea cero.

- ***T1PR:*** Se configura el registro de periodo del Timer 1 para generar una señal de 80 kHz (12.5 μs). Para ello se emplea la expresión:

$$TxPR = \frac{T_{PWM}}{T_{clock} \cdot TPS} - 1$$

  Teniendo en cuente que el reloj de la CPU es de 50 ns y se emplea un prescaler de 1, para generar una señal de 12.5 μs es necesario realizar: 12.5 μs /(50 ns * 1) = 250 cuentas en el temporizador.

- ***T1CNT:*** Se almacena el valor inicial para el contador.

- ***EVIMRA:*** Se procede a habilitar la interrupción por comparación en el Timer 1 (T1CINT, bit 8). Los demás registros *EVIMRx* se ponen a cero al no emplearse más interrupciones del gestor de eventos.

- ***EVIFRA:*** Por último se borran todas las posibles interrupciones en espera en todos los registros *EVIFRx*, escribiendo en todos sus bits un '1'.

Para almacenar los datos que van a ser escritos en el registro de comparación se emplean dos tablas iguales, las cuales serán utilizadas de manera alternativa, es decir, cuando se hayan empleado todos los valores de una de ellas, se comienza a emplear la otra. El cambio de una a otra se realiza en la rutina de interrupción, al igual que el cambio del valor del registro de comparación. Al final de dicha rutina siempre debe borrarse el flag indicador de la interrupción atendida, en este caso en el registro EVIFRA. El resto del programa principal tiene la misma estructura que en ejemplos anteriores, a excepción del empleo de variables para la identificación e indexado de las dos tablas y sus elementos. Los archivos empleados para este proyecto son:

- ***pwm04.c:*** Fichero principal que contiene el código en C para este ejemplo. En él se añaden registros de configuración de interrupciones y del temporizador de usos general GPT1

- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.

- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.

- ***pwm01.cmd:*** Fichero empleado por el enlazador (linker) para la ubicación de variables y constantes en la memoria del DSP (idéntico a los anteriores).

- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.

- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

- **Fichero pwm04.c:**

```
/*****************************************************************/
/* Fichero: pwm04.c                                              */
/* Programa para test de señales PWM                             */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz      */
/*****************************************************************/
/* Generación de señales PWM con temporizador 1 en modo asimétrico */
/* en pin T1PWM/T1CMPR , con polaridad: activa a nivel bajo      */
/* Cambio del duty cycle al vuelo mediante interrupción          */
/* por comparación en Timer 1                                    */
/*****************************************************************/

#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */
```

```
#define LENGTH 16          /* longitud de tabla para variación de duty cycle */
#define PERIOD 250         /* Periodo de la señal PWM*/

/*********************   SETUP del registro OCRA   **********************/
#define OCRA15                  0       /* 0 : IOPB7   1 : TCLKIN              */
#define OCRA14                  0       /* 0 : IOPB6   1 : TDIR                */
#define OCRA13                  0       /* 0 : IOPB5   1 : T2PWM               */
#define OCRA12                  1       /* 0 : IOPB4   1 : T1PWM               */
#define OCRA11                  0       /* 0 : IOPB3   1 : PWM6                */
#define OCRA10                  0       /* 0 : IOPB2   1 : PWM5                */
#define OCRA9                   0       /* 0 : IOPB1   1 : PWM4                */
#define OCRA8                   0       /* 0 : IOPB0   1 : PWM3                */
#define OCRA7                   0       /* 0 : IOPA7   1 : PWM2                */
#define OCRA6                   0       /* 0 : IOPA6   1 : PWM1                */
#define OCRA5                   0       /* 0 : IOPA5   1 : CAP3                */
#define OCRA4                   0       /* 0 : IOPA4   1 :CAP2/QEP2            */
#define OCRA3                   0       /* 0 : IOPA3   1 : CAP1/QEP1           */
#define OCRA2                   0       /* 0 : IOPA2   1 :XINT1                */
#define OCRA1                   0       /* 0 : IOPA1   1 :SCIRXD               */
#define OCRA0                   0       /* 0 : IOPA0   1 : SCITXD              */
/*********************************************************************/
/*********************   SETUP del registro OCRB   **********************/
#define OCRB9                   0       /* 0 : IOPD1   1 : XINT2/EXTSOC        */
#define OCRB8                   1       /* 0 : CKLKOUT  1 : IOPD0              */
#define OCRB7                   0       /* 0 : IOPC7   1 : CANRX               */
#define OCRB6                   0       /* 0 : IOPC6   1 : CANTX               */
#define OCRB5                   0       /* 0 : IOPC5   1 : SPISTE              */
#define OCRB4                   0       /* 0 : IOPC4   1 : SPICLK              */
#define OCRB3                   0       /* 0 : IOPC3   1 : SPISOMI             */
#define OCRB2                   0       /* 0 : IOPC2   1 : SPISIMO             */
#define OCRB1                   1       /* 0 : BIO     1 : IOPC1               */
#define OCRB0                   1       /* 0 : XF      1 : IOPC0               */
/*********************************************************************/
/*********************   SETUP del WDCR   ********************************/
#define WDDIS                   1       /* 0 : Watchdog enabled 1: disabled    */
#define WDCHK2                  1       /* 0 : System reset   1: Normal OP     */
#define WDCHK1                  0       /* 0 : Normal Oper.     1: sys reset   */
#define WDCHK0                  1       /* 0 : System reset   1: Normal OP     */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64       */
/*********************************************************************/
/*********************   SETUP del registro SCSR   **********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                  */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE     */
#define ILLADR                  1       /* 1 : borrar ILLADR                   */
/*********************************************************************/
/*********************   SETUP de WSGR   ********************************/
#define BVIS            0       /* 10-9 : 00, Bus visibility OFF             */
#define ISWS            0       /* 8 -6 : 000, 0 estados de espera para I/O  */
#define DSWS            0       /* 5 -3 : 000, 0 estados de espera para  datos */
#define PSWS            0       /* 2 -0 : 000, 0 estados de espera para código */
/*********************************************************************/
```

```
/******************      SETUP del registro GPTCON    *********************/
#define GPTCON_T2TOADC        0
                              /* 10-9  : T2TOADC = 00 : GPT2 no inicia ADC          */
#define GPTCON_T1TOADC        0
                              /*  8-7  : T1TOADC = 00 : GPT1 no inicia ADC          */
#define GPTCON_TCOMPOE        1
            /*   6   : TCOMPOE = 1  : habilitadas las salidas de comparación del GPT */
#define GPTCON_T2PIN          0
                        /*  3-2  : T2PIN = 00 : Pol. de salida GPT2 = forzada nivel bajo */
#define GPTCON_T1PIN          1
                         /*  1-0  : T1PIN = 01 : Pol. de salida GPT1 = activa nivel bajo */
/*************************************************************************/
/*********************      SETUP del registro T1CON    *********************/
#define T1CON_FREESOFT        0
                      /* 15-14 FREE, SOFT : 00 stop en emulación JTAG suspendido    */
#define T1CON_TMODE           2
                           /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T1CON_TPS             0
                  /* 10-8  : TPS2-0     : 000  prescaler del reloj de entrada CPUCLK/1  */
#define T1CON_TENABLE         1      /*  6    : TENABLE    : 1 habilita GPT1      */
#define T1CON_TCLKS           0
                                  /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T1ON_TCLD             1
                        /* 3-2   : TCLD1,0    : 01 Recarga del timer 1 al llegar a 0 o a T */
#define T1CON_TECMPR          1
                /* 1     :  TECMPR     : 1 habilita operación de comparación del timer     */
/*************************************************************************/
/*********************    SETUP del registro EVIMRA    *********************/
#define T1OFINT               0  /* 10 : Timer 1 overflow interrupt                */
#define T1UFINT               0  /*  9 : Timer 1 underflow interrupt               */
#define T1CINT                1  /*  8 : Timer 1 compare interrupt                 */
#define T1PINT                0  /*  7 : Timer 1 period interrupt                  */
#defineCMP3INT                0  /*  3 : Compare 3 interrupt                       */
#define CMP2INT               0  /*  2 : Compare 2 interrupt                       */
#define CMP1INT               0  /*  1 : Compare 1 interrupt                       */
#define PDPINT                0  /*  0 : Power Drive Protect Interrupt             */
/*************************************************************************/

/*********************    SETUP del registro EVIMRB    *********************/
#define T2OFINT               0  /*  3 : Timer 2 overflow interrupt                */
#define T2UFINT               0  /*  2 : Timer 2 underflow interrupt               */
#define T2CINT                0  /*  1 : Timer 2 compare interrupt                 */
#define T2PINT                0  /*  0 : Timer 2 period interrupt                  */
/*************************************************************************/

/*********************    SETUP del registro EVIMRC    *********************/
#define CAP3INT               0  /*  2 : Capture Unit 3 interrupt                  */
#define CAP2INT               0  /*  1 : Capture Unit 2 Interrupt                  */
#define CAP1INT               0  /*  0 : Capture unit 1 interrupt                  */
/*************************************************************************/
```

```
/**********************    SETUP del registro IMR    **********************/
#define INT6                  0  /*  5 : Level INT6  enmascarada                */
#define INT5                  0  /*  4 : Level INT5  enmascarada                */
#define INT4                  0  /*  3 : Level INT4  enmascarada                */
#define INT3                  0  /*  2 : Level INT3  enmascarada                */
#define INT2                  1  /*  1 : Level INT2  no enmascarada             */
#define INT1                  0  /*  0 : Level INT1  enmascarada                */
/*****************************************************************************/

interrupt void ser(void);       /* declaración de rutina de interrupción */
extern _out_wsgr();            /* declaración de función externa para configurar WSGR */

unsigned char RepIsrNo;          /* Indicador de posición dentro de la tabla actual */
unsigned char lookupTable;       /* Número de la tabla actual: 0 = tabla1; 1 = tabla2 */
unsigned int *pt;                /* Puntero a elemento dentro de la tabla actual */

/* Tablas de valores para el test de cambio alternativo del duty cylce */

unsigned int table1[16]={40,1,40,1,40,1,40,1,40,1,40,1,40,1,40,1};
unsigned int table2[16]={40,1,40,1,40,1,40,1,40,1,40,1,40,1,40,1};

interrupt void ser(void)        /* Rutina de interrupción para cambio del duty cycle */
{
 if((PIVR-0x0028)==0)       /* Verifica origen de la interrupción  (28=TCINT1)*/
        {
         if(lookupTable==0 && RepIsrNo==LENGTH)
                {
                pt=table2;                      /* Si es final de tabla 1, pasa a tabla 2 */
                lookupTable=1;
                RepIsrNo=0;
                }

        if(lookupTable==1 && RepIsrNo==LENGTH)
                {
                pt=table1;                       /* Si es final de tabla 2, pasa a tabla 1 */
                 lookupTable=0;
                RepIsrNo=0;
                }
        T1CMPR=*pt++;               /* Carga próximo valor de la tabla en T1CMPR*/
        RepIsrNo++;                 /* Incrementa el indicador en la tabla */
        EVIFRA=(T1CINT<<8);     /* Borra flag de interrupción T1CINT */
        }
}

void c_dummy1(void)
{
        while(1);               /* función para atrapar interrupciones espúreas */
}
```

```
/**********************    Programa principal MAIN  ***********************/

void main(void)
{
asm (" setc INTM");          /* Deshabilita todas las interrupciones */
asm (" clrc SXM");           /* Borra el bit de extensión de signo */
asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                                  /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);         /* Inicializa SCSR     */

out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                        /* Función externa para configurar WSGR */

 OCRB = ((OCRB9<<9)+(OCRB8<<8)+
                (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
                (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                          /* Inicializa OCRB */

 OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                          /* Inicializa OCRA */

GPTCON=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
            (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                                      /* Inicia el registro de control del temporizador */

T1PR = PERIOD;               /* Inicia periodo del Timer 1*/
lookupTable=0;
pt=table1;                   /* Puntero a la tabla 1 */
T1CMPR = *pt++;              /* Inicia valor del comparador con dato de tabla */
RepIsrNo=1;                  /* fuera de la rutina de interrupción*/
T1CNT=0x0000;                /* Valor inicial del contador del T1 */

T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+(T1CON_TPS<<8)+
                (T1CON_TCLKS<<4)+(T1CON_TCLD<<2)+(T1CON_TECMPR<<1));
                                    /* Configura el registro de control del temporizador 1 */

EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
                        (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                                          /* Registro de máscaras del grupo A del EV  */

 EVIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                                          /* Registro de máscaras del grupo B del EV  */
```

```
EVIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                                        /* Registro de máscaras del grupo C del EV  */

EVIFRA=0xFFFF;      /* Registro de flags del grupo A del EV  */

EVIFRB=0xFFFF;      /* Registro de flags del grupo B del EV  */

EVIFRC=0xFFFF;      /* Registro de flags del grupo C del EV  */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                                        /* Registro de máscaras de interrupción */

IFR=0xFFFF;             /* Borra todas las interrupciones pendientes */

asm (" clrc INTM");             /* Habilita todas las interrupciones */

T1CON=T1CON+(T1CON_TENABLE<<6);                 /* Habilita el GPT1*/

while(1);               /* Bucle sin fin. Todo lo realiza la rutina de interrupción */
}
```

La estructura del archivo principal es exactamente la misma que en ejemplos anteriores. Únicamente varían las líneas en las que se definen las tablas y variables para su empleo (pt, lookuptable, RepIdrNo, table1 y table2), además de la rutina de interrupción (ser). El puntero "pt" se encarga de indicar el elemento a tomar de la tabla correspondiente en cada caso; la variable "lookuptable" es un elemento auxiliar para saber qué tabla es la actual; la variable "RepIsrNo" almacena el número de datos que han sido empleados de la tabla actual. Con estos elementos se comprueba continuamente en la rutina de interrupción en qué elemento y tabla nos encontramos en cada momento, por si es necesario continuar con la otra tabla de valores. Los valores "40" y "1" empleados en las tablas se han elegido como pruebas, ya que se podían haber tomado cualquier grupo de valores entre 0 y 250 (periodo). Al ser la señal activa a nivel bajo y asimétrica, mientras menor sea el valor en el registro de comparación más tiempo estará la señal a cero, ya que al aumentar el valor del comparador únicamente actuamos sobre el flanco de bajada de la señal, llevándolo desde 0 hasta el periodo (figura 4.52).

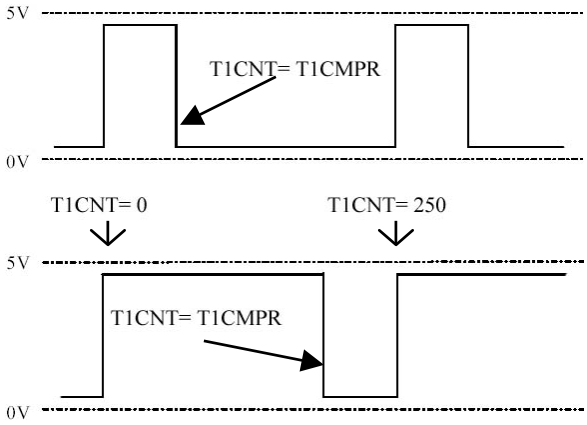


Figura 4.53. Variación del ancho de pulso (activo nivel bajo).

Si la señal fuese activa a nivel alto, se actuaría sobre el flanco se subida, con lo que a mayor valor del comparador más tiempo está la señal a cero. Esta es una de las características de las señales PWM asimétricas: que solo puede controlarse sobre uno de los flancos de las señal.

- **Fichero vectors.asm:**

```
        .title  "vectors.asm"
        .ref    _c_int0,_c_dummy1,_ser
        .sect   ".vectors"

reset:      b       _c_int0
int1:       b       _c_dummy1
int2:       b       _ser
int3:       b       _c_dummy1
int4:       b       _c_dummy1
int5:       b       _c_dummy1
int6:       b       _c_dummy1
reserved:   b       _c_dummy1
sw_int8:    b       _c_dummy1
sw_int9:    b       _c_dummy1
sw_int10:   b       _c_dummy1
sw_int11:   b       _c_dummy1
sw_int12:   b       _c_dummy1
sw_int13:   b       _c_dummy1
sw_int14:   b       _c_dummy1
sw_int15:   b       _c_dummy1
sw_int16:   b       _c_dummy1
trap:       b       _c_dummy1
nmint:      b       _c_dummy1
emu_trap:   b       _c_dummy1
sw_int20:   b       _c_dummy1
sw_int21:   b       _c_dummy1
sw_int22:   b       _c_dummy1
sw_int23:   b       _c_dummy1
```

La tabla de vectores de interrupción es igual que las empleadas anteriormente, salvo que se incluye en la interrupción 2 la llamada a la rutina de interrupción “ser” definida en el programa principal. De esta forma se asocia esa rutina de interrupción a uno de los niveles de interrupción de la CPU.

4.6.7.4. Ejemplo 6:

Este nuevo ejemplo pretende mostrar el uso de las señales PWM del procesador como convertidor Digital-Analógico, además de otro modo de emplear las interrupciones de un temporizador para programar tiempos de muestreo. Su función consiste en generar una onda Seno y otra Coseno de 340 Hz a partir de ondas PWM de 10.8 kHz y duty cycle variable en los pines de salida T1PWM y T2PWM. Éste ejemplo está diseñado para emplearse con la placa que se empleará más adelante para el control de motores, ya que en ella dichas salidas están provistas de filtros de paso bajo de 1 kHz con las que filtrar las señales PWM y mostrar en un osciloscopio las señales Seno y Coseno. Para generar ambas señales se emplean ondas PWM asimétricas de frecuencia constante y ancho de pulso variable, lo cual se consigue cargando en el registro comparador de cada temporizador los datos de comparación contenidos en tablas con el orden adecuado. Las señales PWM se generan con los temporizadores 1 y 2 que serán activos a nivel alto, y estarán sincronizados en su inicio al emplear el Timer 2 el bit ENABLE del Timer 1. Los valores de comparación almacenados en las tablas están ordenados de manera adecuada para que en cada caso la onda generada sea Seno o Coseno, es decir, los datos están desfasados 90º.

Este ejemplo se basa en el hecho de que la señal PWM se puede descomponer en una componente de continua más una señal cuadrada de valor medio cero, de forma que al variar el ancho del pulso se modifica el valor de la componente de continua. De esta forma y junto con los filtros de paso bajo, se obtienen ondas PWM filtradas que resultan ser un Seno o un Coseno. Para ello, al ser las salidas activas a nivel alto, cuando el registro comparador tenga el valor del periodo (onda todo el periodo a cero menos en un instante al final del mismo) la onda generada tiene su componente de continua con valor mínimo (0V); si el valor es la mitad del periodo (duty cycle al 50%) la onda generada se encuentra en su valor medio (2.5V); y si el comparador vale cero (onda activa todo el periodo)se encuentra en su valor medio máximo (5V). De esta forma se obtienen las ondas Seno y Coseno al obtener tensiones variables entre 0 y 5V cambiando el ancho del pulso de las señales PWM.

La estructura de este ejemplo sigue con lo detallado en ejemplos anteriores, por lo que los archivos necesarios siguen siendo los mismos. Hay pocas variaciones respecto a los ejemplos anteriores. Una de ellas es que se emplean las interrupciones como comprobación de que ha ocurrido un evento, pero no van a detener el programa para atender a una rutina de interrupción. Por ello el archivo “vector.asm” no presenta ningún salto a rutina como en casos anteriores. En el programa principal las modificaciones que presentan se deben a la necesidad de configurar los dos temporizadores de uso general del DSP: habilitar la comparación de ambos y sus salidas, habilitarlos con el mismo bit ENABLE y establecer la máscara de interrupción por periodo del Timer 1. Otra característica que presente es la comprobación en bucle infinito del flag de interrupción por periodo del temporizador 1, lo cual se emplea como condición para cambiar el valor del registro comparador en ambos temporizadores. De esta manera cada valor de la tabla se emplea un tiempo igual al periodo de la señal PWM, 10.8 kHz (92 μs), por lo que al emplear 32 valores en la tabla la onda resultante es de una frecuencia de 340 Hz (92 μs * 32 = 2.9 ms). Como siempre, el flag de interrupción debe ser borrado para permitir el próximo cambio de valor del comparador. También hay que introducir código que se encargue de volver al inicio de la tabla cuando esta se acabe.

Los archivos empleados para este proyecto son:

- ***sine.c:*** Fichero principal que contiene el código en C para este ejemplo.
- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.
- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.
- ***F243DIL.cmd:*** Fichero empleado por el enlazador (linker) para la ubicación de variables y constantes en la memoria del DSP.
- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.
- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

- **Fichero sine.c:**

```
/*********************************************************************/
/* Fichero: sine.c                                                   */
/* Programa para el test de señales PWM                              */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz          */
/*********************************************************************/
/* Generación de señales seno y coseno a partir de unas tablas       */
/* en pines T1PWM y T2PWM , con polaridad: activa a nivel alto       */
/* Emplea señales PWM en T1PWM y T2PWM de 10.8 kHz  (92 µs)          */
/* para generar ondas seno y coseno de 340 Hz                        */
/*********************************************************************/
#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */
#define PERIOD 921  /* Periodo de la señal PWM = 92 µs/(50 ns * 2) */
/*********************  SETUP del registro OCRA  **********************/
#define OCRA15              0       /* 0 : IOPB7   1 : TCLKIN        */
#define OCRA14              0       /* 0 : IOPB6   1 : TDIR          */
#define OCRA13              1       /* 0 : IOPB5   1 : T2PWM         */
#define OCRA12              1       /* 0 : IOPB4   1 : T1PWM         */
#define OCRA11              0       /* 0 : IOPB3   1 : PWM6          */
#define OCRA10              0       /* 0 : IOPB2   1 : PWM5          */
#define OCRA9               0       /* 0 : IOPB1   1 : PWM4          */
#define OCRA8               0       /* 0 : IOPB0   1 : PWM3          */
#define OCRA7               0       /* 0 : IOPA7   1 : PWM2          */
#define OCRA6               0       /* 0 : IOPA6   1 : PWM1          */
#define OCRA5               0       /* 0 : IOPA5   1 : CAP3          */
#define OCRA4               0       /* 0 : IOPA4   1 :CAP2/QEP2      */
#define OCRA3               0       /* 0 : IOPA3   1 : CAP1/QEP1     */
#define OCRA2               0       /* 0 : IOPA2   1 :XINT1          */
#define OCRA1               0       /* 0 : IOPA1   1 :SCIRXD         */
#define OCRA0               0       /* 0 : IOPA0   1 : SCITXD        */
/*********************************************************************/
```

```
/*********************     SETUP del registro OCRB     *********************/
#define OCRB9                   0       /* 0 : IOPD1   1 : XINT2/EXTSOC      */
#define OCRB8                   1       /* 0 : CKLKOUT  1 : IOPD0            */
#define OCRB7                   0       /* 0 : IOPC7   1 : CANRX             */
#define OCRB6                   0       /* 0 : IOPC6   1 : CANTX             */
#define OCRB5                   0       /* 0 : IOPC5   1 : SPISTE            */
#define OCRB4                   0       /* 0 : IOPC4   1 : SPICLK            */
#define OCRB3                   0       /* 0 : IOPC3   1 : SPISOMI           */
#define OCRB2                   0       /* 0 : IOPC2   1 : SPISIMO           */
#define OCRB1                   1       /* 0 : BIO     1 : IOPC1             */
#define OCRB0                   1       /* 0 : XF      1 : IOPC0             */
/*****************************************************************************/

/*********************     SETUP del WDCR     ***********************/
#define WDDIS                   1       /* 0 : Watchdog enabled 1: disabled  */
#define WDCHK2                  1       /* 0 : System reset   1: Normal OP   */
#define WDCHK1                  0       /* 0 : Normal Oper.     1: sys reset */
#define WDCHK0                  1       /* 0 : System reset   1: Normal OP   */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64     */
/*****************************************************************************/

/*********************     SETUP del registro SCSR     *********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE   */
#define ILLADR                  1       /* 1 : borrar ILLADR                 */
/*****************************************************************************/

/************************     SETUP de WSGR     ************************/
#define BVIS            0       /* 10-9 : 00, Bus visibility OFF             */
#define ISWS            0       /* 8 -6 : 000, 0 estados de espera para I/O  */
#define DSWS            0       /* 5 -3 : 000, 0 estados de espera para  datos */
#define PSWS            0       /* 2 -0 : 000, 0 estados de espera para código */
/*****************************************************************************/

/*******************     SETUP del registro GPTCON     *********************/
#define GPTCON_T2TOADC      0
                        /* 10-9  : T2TOADC = 00 : GPT2 no inicia ADC         */
#define GPTCON_T1TOADC      0
                        /*  8-7  : T1TOADC = 00 : GPT1 no inicia ADC         */
#define GPTCON_TCOMPOE      1
            /*   6   : TCOMPOE = 1  : habilitadas las salidas de comparación del GPT */
#define GPTCON_T2PIN            0
                    /*  3-2  : T2PIN = 10 : Pol. de salida GPT2 = activa nivel alto */
#define GPTCON_T1PIN            1
                    /*  1-0  : T1PIN = 10 : Pol. de salida GPT1 = activa nivel alto */
/*****************************************************************************/
```

```
/*********************     SETUP del registro T1CON    *********************/
#define T1CON_FREESOFT          2
                    /* 15-14 FREE, SOFT : 10 stop en emulación JTAG no inflye   */
#define T1CON_TMODE             2
                        /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T1CON_TPS               1
                  /* 10-8  : TPS2-0     : 001  prescaler del reloj de entrada CPUCLK/2  */
#define T1CON_TENABLE           1       /*  6     : TENABLE    : 1 habilita GPT1      */
#define T1CON_TCLKS             0
                                /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T1ON_TCLD               1
                        /* 3-2   : TCLD1,0    : 01 Recarga del timer 1 al llegar a 0 o a T */
#define T1CON_TECMPR            1
                /* 1     :  TECMPR     : 1 habilita operación de comparación del timer     */
/*****************************************************************************/

/*********************     SETUP del registro T2CON    *********************/
#define T2CON_FREESOFT
                    /* 15-14 FREE, SOFT : 01 stop en emulación JTAG no inflye   */
#define T2CON_TMODE             2
                        /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T2CON_TPS               1
                  /* 10-8  : TPS2-0     : 001  prescaler del reloj de entrada CPUCLK/2  */
#define T2CON_TSWT1             1   /*  7: TSWT1: 1 usa bit TENABLE de GPT1 */
#define T2CON_TCLKS             0
                                /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T2CON_TCLD              1
                        /* 3-2   : TCLD1,0    : 01 Recarga del timer 2 al llegar a 0 o a T */
#define T2CON_TECMPR            1
                /* 1     :  TECMPR     : 1 habilita operación de comparación del timer     */
#define T2CON_SELT1PR           0
                        /* 0       :  SELT1PR    :0      use su propio registro de periodo  */
/*****************************************************************************/

/*********************    SETUP del registro EVIMRA    *********************/
#define T1OFINT                 0  /* 10 : Timer 1 overflow interrupt              */
#define T1UFINT                 0  /*  9 : Timer 1 underflow interrupt             */
#define T1CINT                  0  /*  8 : Timer 1 compare interrupt               */
#define T1PINT                  1  /*  7 : Timer 1 period interrupt                */
#defineCMP3INT                  0  /*  3 : Compare 3 interrupt                     */
#define CMP2INT                 0  /*  2 : Compare 2 interrupt                     */
#define CMP1INT                 0  /*  1 : Compare 1 interrupt                     */
#define PDPINT                  0  /*  0 : Power Drive Protect Interrupt           */
/*****************************************************************************/

/*********************    SETUP del registro EVIMRB    *********************/
#define T2OFINT                 0  /*  3 : Timer 2 overflow interrupt              */
#define T2UFINT                 0  /*  2 : Timer 2 underflow interrupt             */
#define T2CINT                  0  /*  1 : Timer 2 compare interrupt               */
#define T2PINT                  0  /*  0 : Timer 2 period interrupt                */
/*****************************************************************************/
```

```
/*******************    SETUP del registro EVIMRC    ********************/
#define CAP3INT                 0  /*  2 : Capture Unit 3 interrupt                */
#define CAP2INT                 0  /*  1 : Capture Unit 2 Interrupt                */
#define CAP1INT                 0  /*  0 : Capture unit 1 interrupt                */
/*****************************************************************************/

/**********************    SETUP del registro IMR    ***********************/
#define INT6                0  /*  5 : Level INT6  enmascarada                    */
#define INT5                0  /*  4 : Level INT5  enmascarada                    */
#define INT4                0  /*  3 : Level INT4  enmascarada                    */
#define INT3                0  /*  2 : Level INT3  enmascarada                    */
#define INT2                0  /*  1 : Level INT2  enmascarada                    */
#define INT1                0  /*  0 : Level INT1  enmascarada                    */
/*****************************************************************************/

unsigned int sine[32] = {460, 550, 637, 716, 786, 843, 886, 912,
                         921, 912, 886, 843, 786, 716, 637, 550,
                         460, 371, 284, 205, 135, 78, 35, 9,
                         0, 9, 35, 78, 135, 205, 284, 371 };          /* Tabla de seno */

unsigned int cosine[32] = {921, 912, 886, 843, 786, 716, 637, 550,
                           460, 371, 284, 205, 135, 78, 35, 9,
                           0, 9, 35, 78, 135, 205, 284, 371,
                           460, 550, 637, 716, 786, 843, 886, 912};     /* Tabla de  coseno */

unsigned int x;          /* Variable de uso general */

/**********************    Programa principal MAIN  ***********************/

void main(void)
{
asm (" setc INTM");          /* Deshabilita todas las interrupciones */
asm (" clrc SXM");           /* Borra el bit de extensión de signo */
asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                                  /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);         /* Inicializa SCSR     */

out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                      /* Función externa para configurar WSGR */

 OCRB = ((OCRB9<<9)+(OCRB8<<8)+
                 (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
                 (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                    /* Inicializa OCRB */
```

```
OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                    /* Inicializa OCRA */

GPTCON=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
            (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                                    /* Inicia el registro de control del temporizador */

T1PR = PERIOD;              /* Inicia periodo del Timer 1*/
T1CMPR = PERIOD/2;          /* Inicia valor del comparador */
T1CNT=0xFFFE;               /* Valor inicial del contador del T1 a –2 */

T2PR = PERIOD;              /* Inicia periodo del Timer 2*/
T2CMPR = PERIOD/2;          /* Inicia valor del comparador */
T2CNT=0xFFFE;               /* Valor inicial del contador del T2  a –2 */

T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+(T1CON_TPS<<8)+
              (T1CON_TCLKS<<4)+(T1CON_TCLD<<2)+(T1CON_TECMPR<<1));
                                    /* Configura el registro de control del temporizador 1 */

T2CON=((T2CON_FREESOFT<<14)+(T2CON_TMODE<<11)+(T2CON_TPS<<8)+
                (T2CON_TSWT1<<7)+(T2CON_TCLKS<<4)+(T2CON_TCLD<<2)+
                                    (T2CON_TECMPR<<1)+T2CON_SELT1PR);
                                    /* Configura el registro de control del temporizador 2 */

EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
                      (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                                    /* Registro de máscaras del grupo A del EV  */

 EVIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                                    /* Registro de máscaras del grupo B del EV  */

 EVIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                                    /* Registro de máscaras del grupo C del EV  */

 EVIFRA=0xFFFF;      /* Registro de flags del grupo A del EV  */

 EVIFRB=0xFFFF;      /* Registro de flags del grupo B del EV  */

 EVIFRC=0xFFFF;      /* Registro de flags del grupo C del EV  */

 IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                                                    /* Registro de máscaras de interrupción */

 IFR=0xFFFF;             /* Borra todas las interrupciones pendientes */

 asm (" clrc INTM");             /* Habilita todas las interrupciones */

T1CON=T1CON+(T1CON_TENABLE<<6);        /* Habilita el GPT1 y GPT2 */
```

```
x = 0;

 for ( ; ; )                    /* Bucle infinito */
  {
  /* Comprueba si T1 ha producido interrupción por periodo */

  if ( EVAIFRA & 0x0080)
        {
         EVAIFRA = 0x0080;      /* Borra flag de interrupción del T1*/
         T1CMPR = sine[x];      /* Carga nuevos valores del duty cycle de tablas */
         T2CMPR = cosine[x];
         if ( x < 31)           /* Comprueba si ha llegado a fin de tabla */
           x++;                 /* Pasa a siguiente elemento de tabla */
         else
           x = 0;               /* Pasa al primer elemento de tabla */
        }
  }
}
```

4.6.8. Periféricos del Gestor de Eventos con Vissim:

1) Señales PWM:

El programa de desarrollo VisSim está preparado para emplear todos los periféricos detallados anteriormente, cada uno de ellos en forma de bloque. Todos ellos se encuentran en la pestaña "VisSim/DSP" de la barra superior de menús (figura 4.54). En él se encuentran todos los bloques disponibles de los distintos periféricos del DSP. Para el empleo de los temporizadores de uso general para la generación de ondas PWM, en dicho menú aparece el bloque denominado "PWM". Escogiendo éste bloque (figura 4.55) se pueden generar dichas señales indicando el temporizador a emplear, el modo de conteo, el preescalado del reloj y el periodo en número de cuentas. Para ello, pulsando dos veces en el bloque, se abre la ventana de configuración del mismo (figura 4.56), en la que están disponibles las características indicadas.

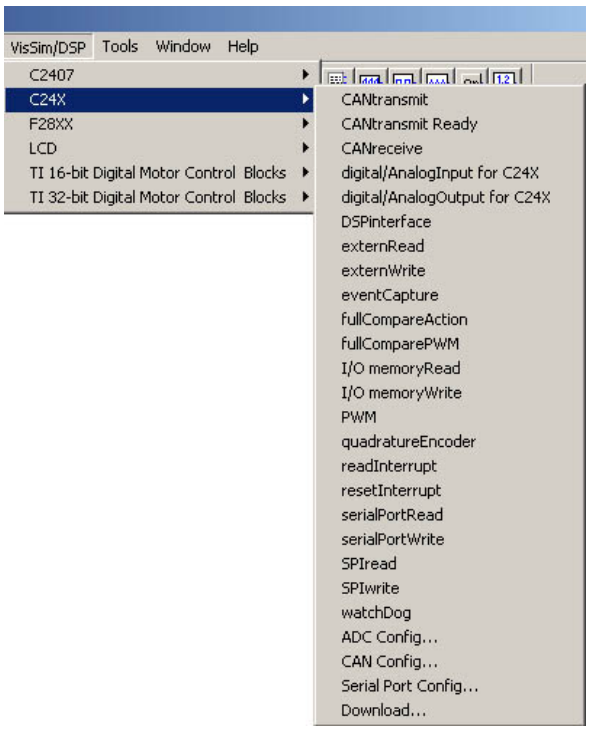


Figura 4.54. Menú VisSim/DSP.

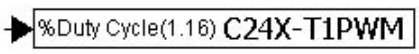


Figura 4.55. Bloque PWM en VisSim.

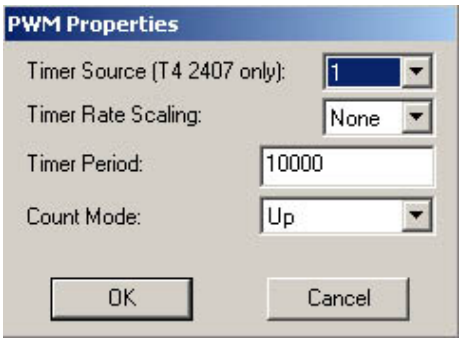


Figura 4.56. Ventana de configuración PWM.

Éste bloque es compartido en el programa para los modelos con el DSP TMS320LF2407, por lo que algunas de sus opciones no son validas para el F243. En particular, la opción "Timer Source" permite seleccionar entre cuatro temporizadores: del 1 al 4. En el 2407 existen estos 4 temporizadores, pero en el F243 sólo existen el 1 y el 2, por lo que el 3 y el 4 no deben seleccionarse para este procesador. En este campo pues se elige el temporizador que se quiere emplear. La siguiente opción, "Timer Rate Scaling", se encarga de seleccionar el preescalado del reloj empleado, que varía entre 1/1 (None) y 1/128, al igual que se analizó anteriormente.. El campo "Timer Period" es para indicar el periodo de la señal generada, lo cual se indica con el número de cuentas que debe realizar el contador como se detalló en apartados anteriores. Por último, el campo "Count Mode" permite seleccionar entre los distintos modos de conteo que disponen los temporizadores: Hold, Up, Up/Down, TDIR Control.

Una vez configuradas las opciones del temporizador, para generar una onda PWM es necesario conectar una entrada al bloque en cuestión. La entrada que permite es el valor del Duty Cycle que se desea emplear pero en tanto por 1, es decir, un valor entre 0 y 1, más concretamente entre 0 y 0.99997 según la referencia técnica del VisSim. Además este dato debe estar en un formato de 1:16 de coma fija, por lo que es necesario emplear los bloques de coma fija situado en el menú superior "Blocks" en la opción "Fixed Point". De esta forma, el valor de la entrada podría ser una constante en formato 1:16 de coma fija empleando la opción "const" del menú anterior, cuyo valor y formato pueden configurarse mediante la ventana de configuración que aparece al pulsar dos veces sobre el bloque situado. También puede emplearse un deslizador o Slider para variar la entrada según se crea conveniente, lo cual se consigue mediante el icono  de la barra superior de menús. En su configuración puede indicarse los valores extremos que va a tomar y el incremento de valor empleado en su variación. Como estos valores deben ser en coma fija, es necesario incluir un bloque de conversión de formato, el cual se encuentra en el menú "Fixed Point" o en el icono  de la barra superior. Éste bloque debe configurarse para formato "Scaled int" y 1:16. De esta forma se podría conseguir el diagrama de la figura 4.57.

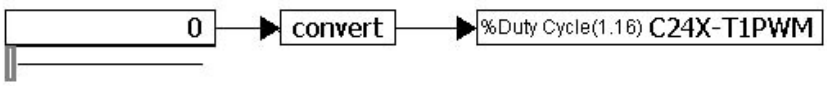


Figura 4.57. Generación de señal PWM con ancho variable.

Hay que destacar que para poder ejecutar estos diagramas en VisSim existen una serie de problemas y restricciones. Una de ellas es el modo en el que se pueden emplear dichos bloques según el temporizador empleado. Si es elegido el temporizador 1, el diagrama puede ejecutarse tal y como está pulsando el botón "GO" para ejecutarlo en modo interactivo. Si se desea, empleando este mismo temporizador puede componerse primero un bloque para compilarlo y generar un archivo ".out" para cargarlo en una interfaz DSP, ejecutando el diagrama en modo compilado. Ambos modos funcionan correctamente con el temporizador 1, pero no es así con el temporizador 2. Debido a problemas no muy aclarados por los desarrolladores del programa, el temporizador 2 únicamente puede emplearse en modo compilado y no en modo interactivo, por lo que debe sustituirse dicho bloque en el diagrama por una interfaz DSP cargada con el archivo ".out" compilado con las opciones adecuadas del temporizador 2.

Otro inconveniente lo presenta el temporizador 1 y es que, al ser empleado también por VisSim para las comunicaciones con el DSP, sólo puede utilizarse con una única configuración de periodo, prescaler y modo de conteo. Los valores son: Prescaler= 1/16, Periodo = 12500 y modo de conteo = Up. Otra configuración distinta a ésta no genera físicamente ninguna señal PWM ni en modo compilado ni interactivo. Debido a esto, sólo pueden generarse ondas PWM con el temporizador 1 de 100 Hz de frecuencia (12500 * 16 * 50ns = 0.01 s → 100 Hz).

También es posible emplear las unidades completas de comparación para generar señales PWM para controlar inversores trifásicos. Para ello basta con elegir el bloque "FullcomparePWM" del menú "Vissim/DSP" de la barra superior de menús (figura 4.54). Las entradas a dicho bloque, al igual que antes, son el tanto por uno que va a estar ON cada una de las 3 señales principales generadas por cada unidad de comparación (figura 4.58), siendo estos datos en coma fija y formato de 1:16. Los datos de este bloque se refieren únicamente a las señales principales, ya que las complementadas las genera a partir de ellas.

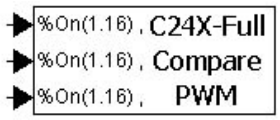


Figura 4.58. Bloque generador PWM con comparadores.

Pulsando con el botón derecho sobre el bloque obtenemos la ventana de configuración para la generación de las señales PWM (figura 4.59). En dicha ventana se muestran los distintos campos para la configuración de los comparadores, entre las que se encuentran las de configuración de la señal PWM que son las mismas que para el bloque explicado anteriormente. De las nuevas opciones, destaca la casilla que permite seleccionar el modo de generación de señales PWM por vectores espaciales "Enable Space Vector Mode", con lo que el programa se encarga de realizar todas las operaciones necesarias para su generación basándose en el resto de datos de configuración. La casilla "External Action Specification", permite controlar de forma externa la acción de este bloque, para lo que añade una señal nueva de entrada denominada "Action" en la que se conecta la señal controladora. En las opciones "PMW Pin" y "Action", se seleccionan para cada señal PWMx la polaridad que deben tener: activas o forzadas a nivel alto o bajo. Por último, mencionar la posibilidad de generar tiempos muertos habilitando la casilla "Use DeadBand", lo que habilita las casillas inferiores para la configuración de la duración de los tiempos muertos.

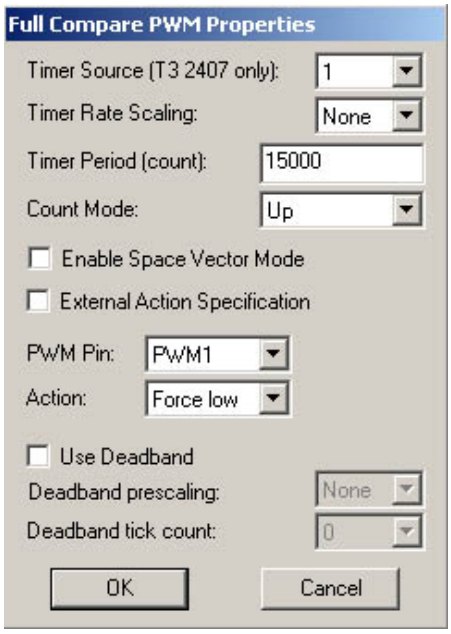


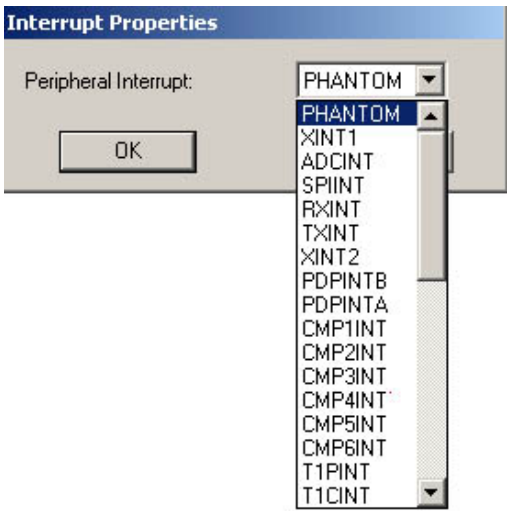
Figura 4.59. Configuración comparadores PWM.

Ejemplos del uso de estos bloques se encuentran en la carpeta "..\VisSim50\dsp\EXAMPLES" de la versión comprada del programa, como por ejemplo el diagrama "bldc2407.vsm".

2) Interrupciones:

VisSim también proporciona la posibilidad de emplear interrupciones en el DSP para el desarrollo de un diagrama. Para ello dispone de dos bloques en el menú "Vissim/DSP" de la barra superior: "InterrupRead" y "InterrupReset" (figura 4.54). El primero de los bloques (figura 4.60) tiene una salida que se activa a nivel alto cuando la interrupción configurada se ha producido, con lo que se puede emplear como señal de control para activar otras aplicaciones del diagrama.

Figura 4.60. Bloques de lectura y Reset de interrupción.



La ventana de configuración permite la elección de la interrupción que quiere comprobarse eligiéndola de una lista desplegable. En dicha lista se encuentran todas la interrupciones que pueden producirse en el DSP como se detalló en la tabla 4.19. También es posible resetear una interrupción, para lo que se dispone del bloque "InterrupReset" (figura 4.60), el cual dispone de una entrada que al activarse a nivel alto resetea la interrupción seleccionada.

Figura 4.61. Selección de interrupción.

3) Señales de captura y QEP:

VisSim dispone de bloques que permiten el muestreo y la captura de transición de señales en determinados pines del dispositivo. Está disponible en el menú "Vissim/DSP" la opción "eventCapture" (figura 4.62).

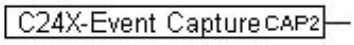


Figura 4.62. Bloque de captura.

Este bloque captura el valor actual del temporizador empleado cuando ocurre la transición seleccionada, por lo que da como salida un intervalo de tiempo entra captura y captura. Éste dato puede emplearse para el cálculo de velocidad mediante un bloque "Speed Calculator".

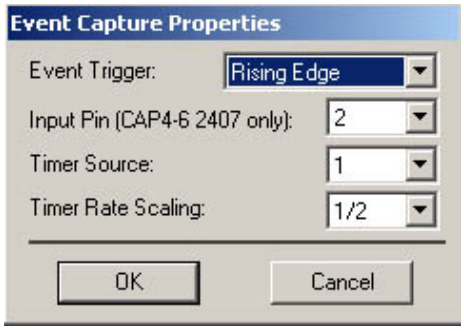


Figura 4.63. Configuración bloque de captura.

La ventana de configuración permite elegir el evento que causa la captura en el campo "Evente Trigger", como son: flanco de subida (Rising Edge), flanco de bajada (Falling Edge= o ambos (Both Edges). También dispone del campo "Input Pin" para la elección del pin a leer. En el TMS320F243 sólo se disponen de las señales CAP1, CAP2 y CAP3, las demás son para otros modelos. Mediante el campo "Timer Source" podemos elegir el temporizador que se empleará para las capturas: Timer 1 o 2. Por último se dispone de la opción de escalar el reloj del temporizador para disminuir su frecuencia mediante el campo "Timer Rate Scaling".

Otro bloque de captura que contiene el VisSim es la de captura de pulsos en cuadratura o QEP, también disponible en el menú "Vissim/DSP" en la opción "QuadratureEncoder" (figura 4.64). con este tipo de capturas se puede calcular al posición y velocidad de una máquina rotatoria.

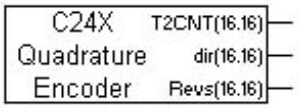


Figura 4.64. Bloque QEP.

Éste bloque dispone de tres salidas: salida "T2CNT" que se corresponde con el contador del temporizador 2, salida "dir" que indica la dirección de giro mediante '+1' si es hacia delante y '-1' si es hacia atrás, y la salida "Revs" que cuenta los pulsos indicadores si está habilitada la casilla "Inc/Dec Rev Count on Index Pulse" en la ventana de configuración del bloque (figura 4.65). En ella además, se pueden seleccionar las entradas del encoder, que para el TMS320F243 son siempre "QEP1-2".

Figura 4.65. Configuración del bloque QEP.

Ejemplos del uso de las señales QEP se encuentran en el directorio "..\VisSim50\dsp\EXAMPLES", como por ejemplo el diagrama "Fan243.vsm", en el que se recoge el uso de los bloques detallados anteriormente para el control de posición de un ventilador mediante señales PWM (figura 4.66).

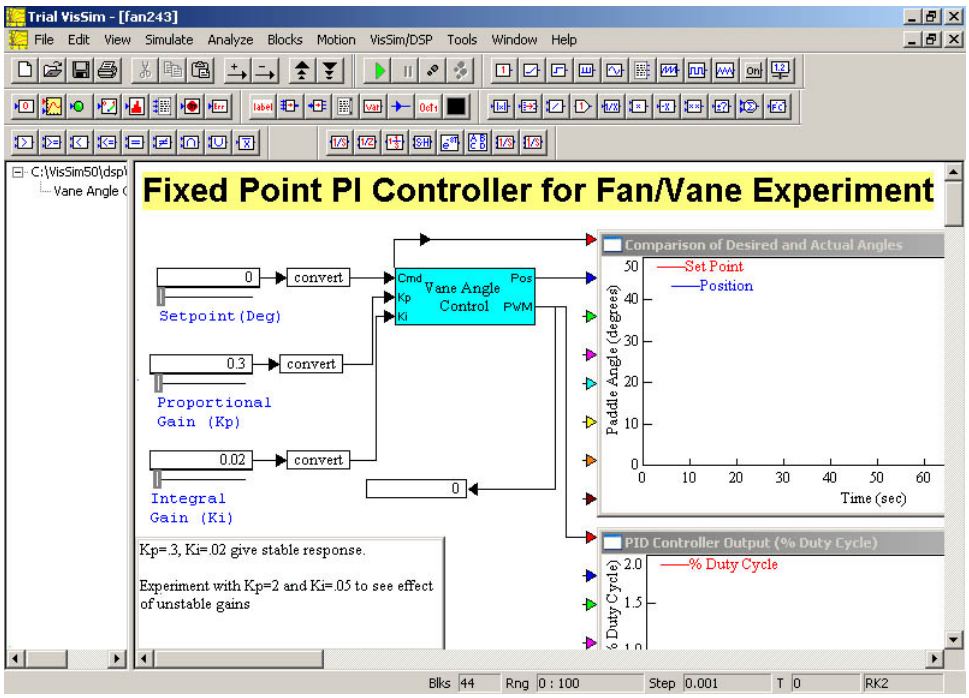


Figura 4.66. Ejemplo "Fan243.vsm".

En este ejemplo existe un bloque compuesto "Vane Angle Control" que pulsando sobre él nos muestra el contenido de dicho bloque (figura 4.67). En ella se muestra el diagrama necesario para el control de posición del ventilador. Ello se realiza a través de la medida del ángulo mediante un bloque QEP, cuyos pulsos se convierten en grados al

emplear un factor de escala adecuado. Conocida la posición del ventilador y la referencia se calcula la acción a tomar mediante un control PI, lo que producirá como resultado el duty cycle necesario de la señal PWM que controla el motor del movimiento.

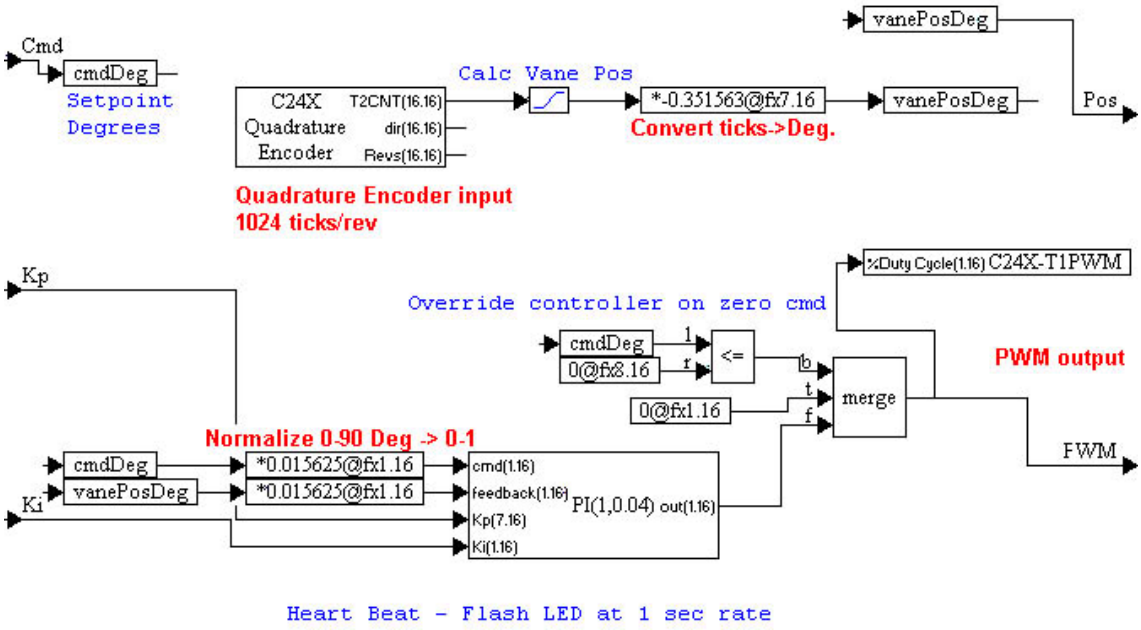


Figura 4.67. Control de posición del ventilador.

4) Watchdog:

Aunque no sea un periférico del Gestor de Eventos, existe también en VisSim un bloque que permite configurar el funcionamiento del temporizador de guarda o Watchdog. Éste bloque se encuentra disponible en la opción "watchdog" del menú "Vissim/DSP" (figura 4.68). Este bloque en principio no tiene entradas ni salidas, pero en su ventana de configuración (figura 4.69) puede habilitarse la casilla "Use Input To Enable" para colocar una señal de entrada que permita activar el watchdog cuando sea necesario.

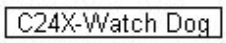


Figura 4.68. Bloque Watchdog.

En la ventana de configuración se puede introducir el valor del tiempo en milisegundos para que el Watchdog proceda a ejecutar un reset del sistema.

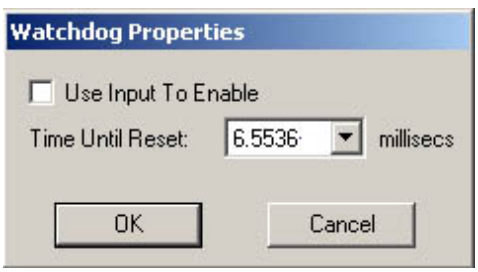


Figura 4.69. Configuración del Watchdog.

4.7. CONVERTIDOR ANALÓGICO-DIGITAL (A/D):

Uno de los periféricos más empleados en los sistemas de control es el convertidor Analógico-Digital, el cual permite transformar una señal analógica en un formato de número digital que resulta más útil para su procesamiento mediante una programación digital. El procesador TMS320F243 de Texas Instruments dispone físicamente de un solo convertidor A/D de 10 bits, pero para tener compatibilidad con otros modelos de la misma empresa se ha añadido una lógica de control que permite utilizarlo como si se tratasen de dos convertidores: ADC1 y ADC2, por lo que la nomenclatura empleada puede confundir al usuario haciéndole creer que son físicamente dos convertidores. Las características principales de este elemento son:

- Convertidor A/D con etapa de muestreo y retención incluido (Sample & Hold).
- Tiempo mínimo de conversión de 850 ns.
- 8 canales de entrada analógica: ADCIN0-ADCIN7.
- Capacidad de realizar dos conversiones de manera casi simultánea.
- Modos de conversión continuo o simple.
- Posibilidad de iniciar la conversión por software, por un evento interno y/o externo.
- Puede suministrarse referencias externas de tensión alta y baja mediante pines externos del dispositivo: VREFHI y VREFLO.
- Registros de dos niveles de profundidad para almacenar el resultado de la conversión.
- Dos registros de control programables: *ADCTRL1* y *ADCTRL2*.
- Prescaler del reloj programable para variar el tiempo de muestreo y retención.
- Capacidad de generar interrupciones programables en prioridad (alta o baja).

El resultado digital de la conversión realizada por el conversor de 10 bits se puede aproximar por la siguiente expresión:

$$Valor\_Digital = 1023 \times \frac{Entrada\_analógica - V_{REFLO}}{V_{REFHI} - V_{REFLO}}$$

donde V_{REFHI} y V_{REFLO} son las tensiones de referencia alta y baja. Éstas referencias pueden ser internas o externas, pero deben cumplir siempre que:

$$0 \le V_{REFLO} \le V_{ADCIN} \le V_{REFHI} \le 5V$$

Para indicar al procesador la procedencia de estas referencias (interna o externa), el Starter Kit DSKF243 dispone de dos un jumper de configuración: JP3 para V_{REFLO} y JP4 para V_{REFHI}. Cuando uno de ellos se encuentra en la posición 1-2, se indica que la tensión de referencia correspondiente es tomada de la misma placa DSK, mientras que en la posición 2-3 se indica que la referencia es externa y se encuentra conectada al pin V_{REFx} correspondiente.

Existe la posibilidad de iniciar la conversión mediante una señal procedente de un pin externo ADCSOC/XINT2/IOPD1 cuando se configura para ello en el registro de control adecuado. Un flanco de subida en el pin ADCSOC iniciaría la conversión.

El convertidor A/D del procesador TMS320F243 dispone de los siguiente modos de funcionamiento:

- Dos canales de entrada pueden ser muestreados y convertidos casi simultáneamente (850 ns cada una).

- Cada pseudo-unidad ADC puede realizar un única conversión o realizar conversiones de manera continua. Si trabaja en modo continuo realizando la conversión de un solo, el convertidor genera un resultado cada 1 μs. Si realiza la conversión de dos canales presenta un resultado cada 1700 ns.

- Cada pseudo-unidad ADC dispone de dos registros FIFO de dos niveles de profundidad, denominados *ADCFIFO1* para la unidad ADC1 y *ADCFIFO2* para la unidad ADC2. Al ser FIFO de dos niveles, los resultados previos se pierden cuando entra un tercer dato en ellos.

- La conversión puede iniciarse por software, por transición en una señal externa en el pin ADCSOC, o por darse algún tipo de evento en el Gestor de Eventos.

- Los registros de control *ADCTRL1* y *ADCTRL2* del periférico tienen asociados registros auxiliares de precarga (Shadowed), por lo que pueden ser escritos en cualquier momento. Con ellos puede indicarse si la conversión se inicia de forma inmediata o al finalizar la anterior conversión.

- Cuando trabaja realizando una única conversión de uno o dos canales (no modo continuo), se establece a ‘1’ el indicador de interrupción correspondiente cuando se acaba la conversión o el par de conversiones, por lo que se realizará la petición de interrupción si está habilitada.

El circuito conversor analógico-digital requiere de un reloj (ADCCLK) con una frecuencia de 20 MHz o menos. La parte analógica de la conversión toma alrededor de 12 ciclos del reloj ADCCLK, pero a veces son necesarios más ciclos de reloj para la sincronización entre el convertidor y la lógica digital de control. Para permitir que este periférico pueda ser usado por dispositivos con otra frecuencia diferente a 20 MHz, existe un prescaler para el reloj que permite configurarlo a una frecuencia menor de 20 MHz. Esto permite emplear el convertidor en modo continuo a mayores tiempos de muestreo, mejorando así la precisión de la medida.

Anteriormente se comentó que el convertidor A/D está preparado para la generación de interrupciones en el procesador. La generación de estas interrupciones depende del modo de trabajo del convertidor: cuando el convertidor trabaja realizando una única conversión de un solo canal, el flag indicador de interrupción es puesto a '1' al final de cada conversión, pero si realiza una única conversión de dos canales de entrada, el flag se establece al final de cada par de conversiones, ya que primero se realiza la del convertidor ADC1 y después la del convertidor ADC2.

El convertidor ADC puede realizar la petición de una interrupción (PIRQ) al controlador de interrupciones de periféricos (PIE). Esta interrupción puede tener una prioridad alta o baja (tabla 4.19), según esté configurado en el bit correspondiente del registro de control *ADCTRL2*. En ambos casos se emplea el mismo vector de interrupción *PIVR*.

En el registro de control *ADCTRL1*, existe un flag indicador de interrupción que es puesto a '1' al final de una conversión. Si el bit que habilita la interrupción en el registro *ADCTRL1* está habilitado, se procede a realizar la petición de interrupción al controlador PIE. Si el bit de habilitación de interrupción no está establecido al final de la conversión pero lo está poco después, y el flag de interrupción no ha sido borrado, entonces se realiza también la petición de interrupción (PIRQ).

4.7.1. Registro ADCTRL1:

El registro de control *ADCTRL1* se encarga de configurar los siguientes aspectos del convertidor A/D:

- Inicio de la conversión.
- Selección de los canales de entrada a convertir.
- Habilitación del módulo ADC.
- Permitir o no las interrupciones del dispositivo.
- Finalización de la conversión y control de la emulación.

Éste registro se encuentra mapeado en la dirección de memoria de datos 7032h y tiene la siguiente estructura:

15	14	13	12	11	10	9	8
Suspend Soft	Suspend Free	ADCIMSTART	ADC2EN	ADC1EN	ADCCONRUN	ADCINTEN	ADCINTFLAG
RW-0	RW-0	RW-0	SRW-0	SRW-0	SRW-0	SRW-0	RC-0

7	6–4	3–1	0
ADCEOC	ADC2CHSEL	ADC1CHSEL	ADCSOC
R-0	SRW-0	SRW-0	SRW-0

Figura 4.70. Registro de control ADCTRL1 (7032h).

Nota: R = acceso a lectura; W = acceso a escritura; C = borrados escribiendo un '1'; S = Shadowed; -0 = valor después del reset.

- ***Bits 15-14: Soft y Free.*** Estos bits determinan que ocurre en el ADC cuando se produce una parada del emulador, por ejemplo debida a un "break-point". Puede indicarse que el periférico continúe su operación o que se detenga cuando acabe con la conversión actual.

  00 Parada inmediata.
  10 Completa la conversión en curso antes de parar.
  X1 Sigue funcionando libremente.

- ***Bit 13: ADCIMSTAR.*** Inicia la conversión de forma inmediata cuando se escribe un '1' en este bit, abortando la conversión actual.

  0 No se ejecuta acción.
  1 Comienza inmediatamente la conversión.

- ***Bit 12: ADC2EN.*** Habilita o deshabilita el pseudo conversor ADC2, pero no inicia conversión. Éste bit tiene doble buffer (Shadowed), por lo que puede ser escrito mientras una conversión previa sigue en progreso, aunque el cambio de este bit no tendrá efecto hasta que la conversión se haya completado.

  0 Convertidor ADC2 deshabilitado. Registro de dato *ADCFIFO2* permanece sin cambios.
  1 Convertidor ADC2 habilitado.

- ***Bit 11: ADC1EN.*** Habilita o deshabilita el pseudo conversor ADC1, pero no inicia conversión. Éste bit tiene doble buffer (Shadowed), por lo que puede ser escrito mientras una conversión previa sigue en progreso, aunque el cambio de este bit no tendrá efecto hasta que la conversión se haya completado.

  0 Convertidor ADC1 deshabilitado. Registro de dato *ADCFIFO1* permanece sin cambios.
  1 Convertidor ADC1 habilitado.

- ***Bit 10: ADCCONRUN.*** Este bit configura al convertidor para su funcionamiento en modo continuo. Puede ser escrito mientras una conversión anterior está en progreso, aunque el cambio de este bit no tendrá efecto hasta que la conversión se haya completado.

  0 No opera en modo continuo (single conversion).
  1 Conversión en modo continuo habilitada.

- ***Bit 9: ADCINTEN.*** Habilita las interrupciones de convertidor. Si este bit está a '1', se realiza la petición de interrupción cuando el bit indicador ADCINTFLG valga '1'. Este bit es borrado en el reset y es de doble buffer (Shadowed). El bit ADCINTFLAG se pone a '1' cuando acaba la conversión, con lo que permite ser comprobado periódicamente (polling), si el bit ADCINTEN lo habilita.

  0 Interrupciones deshabilitadas.
  1 Interrupciones habilitadas.

- ***Bit 8: ADCINTFLAG.*** Bit indicador de interrupción. Se establece a '1' cuando se produce una interrupción por fin de conversión. Es borrado escribiendo un '1' en dicho bit.

  0 No ha ocurrido interrupción alguna.
  1 Ha ocurrido una interrupción.

- ***Bit 7: ADCEOC.*** Este bit se establece a '1' mientras la conversión sigue en progreso. Cuando ésta acaba, este bit es borrado al mismo tiempo que se establece a '1' el bit indicador de interrupción ADCINTFLAG.

  0 Fin de conversión.
  1 Conversión en progreso.

- ***Bits 6-4: ADC2CHSEL.*** Bits para la selección del canal que utilizará el convertidor ADC2. El dato leído de este canal se almacena en ADCFIFO2.

  000 Canal 0 (ADCIN0).
  001 Canal 1 (ADCIN1)
  010 Canal 2 (ADCIN2).
  011 Canal 3 (ADCIN3).
  100 Canal 4 (ADCIN4).
  101 Canal 5 (ADCIN5).
  110 Canal 6 (ADCIN6).
  111 Canal 7 (ADCIN7).

- ***Bits 3-1: ADC2CHSEL.*** Bits para la selección del canal que utilizará el convertidor ADC1. El dato leído de este canal se almacena en ADCFIFO1.

  000 Canal 0 (ADCIN0).
  001 Canal 1 (ADCIN1)
  010 Canal 2 (ADCIN2).
  011 Canal 3 (ADCIN3).
  100 Canal 4 (ADCIN4).
  101 Canal 5 (ADCIN5).
  110 Canal 6 (ADCIN6).
  111 Canal 7 (ADCIN7).

- ***Bit 0: ADCSOC.*** Bit de inicio de conversión. Éste bit es de doble buffer (Shadowed). Al escribir un '1' en él se inicia la conversión tan pronto como finalice la actual.

  0 No tiene acción.
  1 Se inicia la conversión en cuanto acabe la actual.

Una aplicación muy habitual con este registro es la modificación de los bits de selección de los canales a emplear por cada convertidor. Los bits para la selección del canal que tratará el convertidor ADC1 se encuentran entre el bit 1 y 3, mientras que el canal para el ADC2 están entre los bits 4 y 6 (figura 4.71).

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
									ADC2			ADC1			

Figura 4.71. Selección de canales en ADCTRL1.

Para cambiar únicamente los bits dedicados a la selección del canal y mantener los demás bits invariantes, se suele emplear una operación AND con el número adecuado, de forma que los bits que no se quieren modificar son multiplicados por '1' y los que sí se quieren modificar por '0'. De esta manera, para modificar el canal dedicado al convertidor ADC1 se multiplica por el número FFF1h, que deja todos los bits sin modificar excepto los bits 1 a 3 que son borrados. Al ser borrados pueden ser modificados simplemente mediante la suma (OR) del canal que se quiere muestrear. Por ejemplo, si se quiere cambiar al canal 4 para el convertidor ADC1, la operación completa sería:

```
ADCTRL1 = (ADCTRL1 & 0xFFF1) | (0x0004 << 1);
```

donde 0x0004 expresa en hexadecimal el número del canal que se quiere muestrear, el cual debe ser desplazado una posición a la izquierda para situarse en los bits 1 a 3 y poder sumarse en el lugar adecuado con el registro *ADCTRL1*. Para ello se ha usado la estructura "<< n" que deslaza 'n' bits el valor dado hacia la derecha el valor dado.

De igual forma se puede realizar esta operación para el cambio del canal asociado al convertidor ADC2. En este caso la operación AND debe realizarse con el valor FF8Fh, el cual borra los bits 4 a 6 dedicados a indicar el canal para ADC2. De esta forma, si se quiere cambiar al canal 2 para el convertidor ADC2 la operación sería la siguiente:

```
ADCTRL1 = (ADCTRL1 & 0xFF8F) | (0x0002 << 4);
```

donde 0x0002 expresa en hexadecimal el número del canal, el cual debe ser desplazado 4 posiciones hacia la derecha mediante "<< n" para situar el valor en los bits adecuados.

Gracias a que el registro *ADCTRL1* es de doble buffer, estas modificaciones se pueden realizar en cualquier momento, por lo que tendrán validez en cuanto se acabe la conversión que esté en curso. De esta forma pueden muestrearse más de dos canales de entrada analógica a lo largo de una aplicación, cambiando la configuración del registro *ADCTRL1* cuando sea necesario cambiar de canal. Ejemplo de esto se desarrollará con más detalles en el capítulo dedicado al control de motores.

4.7.2. Registro ADCTRL2:

El registro de control ADCTRL2 se encuentra en la dirección de memoria de datos 7034h. En él se selecciona, entre otras cosas, el prescaler del reloj y el modo de conversión, además de mostrar el estado de los registros FIFO de entrada. Su estructura es la siguiente:

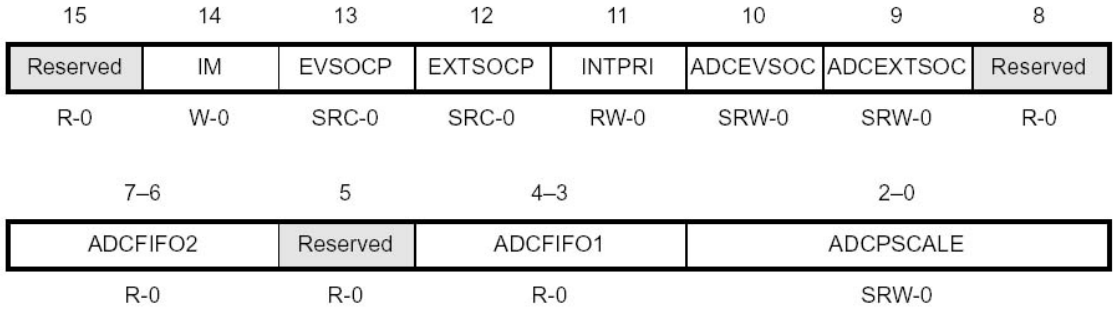


Figura 4.72. Registro de control ADCTRL2 (7034h).

Nota: R = acceso a lectura; W = acceso a escritura; C = borrados escribiendo un '1'; S = Shadowed; -0 = valor después del reset.

- ***Bit 15: Reservado.*** Al leerlo devuelve un cero y la escritura no tiene efecto.

- ***Bit 14: IM.*** Modo de interrupción.

  0 Se establece el flag de interrupción cuando *ADCFIFOx* tiene al menos una palabra almacenada.
  1 Se establece el flag de interrupción sólo cuando *ADCFIFOx* tiene dos palabras almacenadas (resultado).

- ***Bit 13: EVSOCP.*** Inicio de conversión por Gestor de Eventos pendiente. Este bit es de doble buffer (Shadowed). Es borrado mediante la escritura de un '1'. Escribir un '0' no tiene efecto.

  0 No hay conversión pendiente.
  1 Existe conversión pendiente.

- ***Bit 12: EXTSOCP.*** Inicio de conversión por señal externa pendiente. Este bit es de doble buffer (Shadowed). Es borrado mediante la escritura de un '1'. Escribir un '0' no tiene efecto.

  0 No hay conversión pendiente.
  1 Existe conversión pendiente.

- ***Bit 11: INTPRI.*** Prioridad de la interrupción del convertidor A/D. Este bit no es de doble buffer.

  0 Prioridad alta.
  1 Prioridad baja.

- ***Bit 10: ADCEVSOC.*** Bit de habilitación de inicio de conversión por el Gestor de Eventos. El gestor de eventos puede iniciar la conversión por producirse una comparación en registro comparador, por llegada al periodo del contador o por puesta a cero del mismo. Este bit no es de doble buffer.

  0 No tiene acción.
  1 Inicia la conversión A/D por una señal de Gestor de Eventos.

- ***Bit 9: ADCEXTSOC.*** Bit de habilitación de inicio de conversión por señal externa mediante el pin ADCSOC. Este bit no es de doble buffer.

  0 No tiene acción.
  1 Inicia la conversión por una señal procedente del pin ADCSOC.

- ***Bit 8: Reservado.*** Es leído como un cero y su escritura no tiene efecto.

- ***Bits 7-6: ADCFIFO2.*** Bits que muestran el estado de la FIFO de entrada del convertidor ADC2. Dos resultados pueden ser almacenados antes de cualquier lectura. Si después de dos conversiones se realiza una tercera, el resultado más antiguo es perdido. Estos bits no son de doble buffer.

  00 FIFO2 vacía.
  01 FIFO2 tiene un resultado.
  10 FIFO2 tiene dos resultados.
  11 FIFO2 tuvo dos resultados y acaba de recibir otro; el primer valor se ha perdido.

- ***Bit 5: Reservado.*** Es leído como un cero y su escritura no tiene efecto.

- ***Bits 4-3: ADCFIFO1.*** Bits que muestran el estado de la FIFO de entrada del convertidor ADC1. Dos resultados pueden ser almacenados antes de cualquier lectura. Si después de dos conversiones se realiza una tercera, el resultado más antiguo es perdido. Estos bits no son de doble buffer.

  00 FIFO1 vacía.
  01 FIFO1 tiene un resultado.
  10 FIFO1 tiene dos resultados.
  11 FIFO1 tuvo dos resultados y acaba de recibir otro; el primer valor se ha perdido.

- ***Bits 2-0: ADCPSCALE.*** Prescaler del reloj de entrada del convertidor.

  000 CLKOUT/1
  001 CLKOUT/2
  010 CLKOUT/4
  011 CLKOUT/8
  100 CLKOUT/12
  101 CLKOUT/16
  110 CLKOUT/24
  111 CLKOUT/32

4.7.3. Registros ADCFIFO1 y ADCFIFO2:

Los resultados de las conversiones analógico-digitales se almacenan en los 10 bits más significativos de dos registros FIFO de dos niveles: ADCFIFO1 (7036h) y ADCFIFO2 (7038h). Son registros de sólo lectura y son borrados en un reset. Son FIFOs de dos niveles, lo que permite convertir dos variables antes de leerlas de los registros de datos. Sin embargo, un tercer dato provoca la pérdida del valor más antigua almacenado. Su estructura es la siguiente:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Figura 4.73. Registros ADCFIFO1 (7036h) y ADCFIFO2 (7038h).

Nota: R = acceso de lectura; -0 = valor después de reset.

- **Bits 15-6: D9-D0.** Dato convertido y almacenado en los 10 bits más significativos del registro FIFO.

- **Bits 5-0: Reservados.** Siempre se leen a cero.

El modo en el que se almacenan los valores puede parecer extraño, ya que se almacenan en los 10 bits más significativos del registro, al contrario de lo que parecería lógico, almacenarlo en los bits del 9 al 0. Esto es una práctica muy extendida para emplear más cifras decimales de las necesarias, para luego redondear el resultado después de efectuar los cálculos. Añadiendo ceros al final del dato se aumenta la resolución de las futuras operaciones, reduciendo los errores de redondeo. Esto es de gran importancia sobretodo en divisiones, las cuales se producen mediante desplazamientos de los bits hacia la derecha con la consiguiente pérdida de bits.

El valor almacenado en los 10 bits del registro FIFO representan un valor entre 0 y 1023 (3FFh), por lo que para representar una medida en otra escala (de 0 a 5V, 0 a 100%, etc) es necesario multiplicarlo por un factor de escala adecuado. Para obtener el dato del registro FIFO puede procederse de dos manera: extraerse sólo los 10 bits que contienen el dato o emplear el valor total del registro FIFO (todos los bits). En el primer

caso, bastaría con desplazar 6 veces el valor del registro FIFO a la derecha mediante la estructura ">> 6", con lo que se obtendría un valor entre 0 y 1023 (3FFh). En el segundo caso, el valor total leído en el registro FIFO puede tener un valor máximo de FFC0h. Según el caso, la aplicación del factor de escala diferente, puesto que en el segundo caso, al realizar una multiplicación, el resultado será un valor de 32 bits, con lo que al redondear se producen menos errores.

Una vez obtenido el valor del registro se multiplica por el factor de escala adecuado para representar los valores que sean necesarios. Para el caso en el que el valor del registro varíe entre 0 y 1023, el factor de escala puede obtener fácilmente mediante una simple regla de tres, de forma que se asocie el valor máximo del registro (1023) al valor máximo de la escala que se quiere representar. Por ejemplo, si se quiere representar la escala de 0 a 5V se tendría:

$$Factor\_escala = \frac{Máximo\_escala}{1023} = \frac{5V}{1023} = 0.0048875$$

De esta forma, cualquier valor entre 0 y 1023 leído del convertidor, al ser multiplicado por dicho factor de escala, da como resultado un número dentro de la escala requerida. Por ejemplo, si el convertidor ha leído un dato cuyo valor es 1023/2 = 512, al multiplicarlo por el factor de escala se obtiene: 512 * 0.0048875 = 2.5V.

Si el valor obtenido del registro FIFO es mediante lectura de todos los bits, el resultado varía entre 0 y 65472 (FFC0h). El factor de escala puede ser obtenido de una forma semejante a la anterior, pero al multiplicar números de 16 bits el resultado es de 32 bits, debido a que la unidad de multiplicación del DSP contiene un registro de resultado de operación de 32 bits. Trabajando con valores de 32 bits se reducen en gran medida los errores de redondeo, al no emplear factores de escala de pequeño valor y muchos decimales como el caso anterior. Éste procedimiento es más empleado que el anterior debido a las características mencionadas. Si por ejemplo se quieren representar las lecturas en una escala de 0 a 100, se podría proceder multiplicando el valor máximo FFC0h por 100 (64h en hexadecimal), lo que daría como resultado:

```
64h x FFC0h = 63E700h, que redondeando al alza es 640000h.
```

Si se eliminan los ceros de éste valor se obtiene justamente 64h (100 en decimal) como era el objetivo: asociar el valor máximo del registro al valor máximo de la nueva escala. Para eliminar los ceros basta con desplazar el valor 16 posiciones a la derecha, lo que equivale a dividir el resultado de la operación por 2^{16}. De esta forma se extrae el resultado del registro 32 bits de multiplicación, quedándonos con los 16 bits más significativos, con lo que se disminuyen los errores de redondeo. Con todo esto, el valor del factor de escala en decimal puede calcularse mediante la expresión:

$$Factor\_escala = \frac{Máximo\_escala \cdot 2^{16}}{65472}$$

También puede emplearse esta expresión para calcular el factor de escala en hexadecimal si fuese necesario. Basta con sustituir el "máximo de escala" y 65472 por sus valores en dicho formato.

El registro multiplicador trabaja con números con signo, por lo que si los valores muestreados de los canales de entrada no van a tener signo, una práctica común es desplazar el registro FIFO una posición para la derecha, lo que provoca que el bit más significativo sea cero y, por tanto, el valor sea positivo con seguridad. Ello conlleva a que los valores de la FIFO oscilen entre 0 y 7FE0h (32736 en decimal).

Un ejemplo de código en el que se lee un canal mediante el convertidor ADC1 y después se escala el valor obtenido entre 0 y 100, podría ser de la siguiente forma:

```
signed long scaled;         /* define variables empleadas */
signed int value;

scaled = (ADCFIFO1 >> 1);        /* Lee valor de ADC1 y hace
                                          sitio al signo */

scaled* = 0x00C8;     /* Multiplica dato obtenido por factor de
                                        escala adecuado */

value = (signed int)(scaled >> 16);   /* Valor final son los 16
                                       bits más significativos */
```

En este ejemplo se pretende convertir el dato obtenido a la escala 0 a 100, por lo que el factor de escala adecuado se calcula mediante la expresión:

$$Factor\_escala = \frac{100 \cdot 2^{16}}{32736} = 200 \qquad \text{ó} \qquad Factor\_escala = \frac{0x64 \cdot 2^{16}}{0x7FE0} = 0x00C8$$

De esta manera se escalan los datos obtenidos mediante conversiones analógico-digitales para su procesamiento en una aplicación. Existen otros métodos y posibilidades para el manejo de los datos obtenidos de la FIFO, uno de los cuales se describe en el ejemplo que sigue a continuación.

4.7.4. Ejemplos de programación:

4.7.4.1. Ejemplo 7:

El ejemplo que se describe a continuación tiene como objetivo detallar como se puede desarrollar una aplicación en la que se produzcan medidas en canales analógicos con las que realizar algún tipo de operación o algoritmo. Para ello se pondrá en práctica lo expuesto anteriormente sobre configuración y lectura de los dos pseudo-convertidores Analógico-Digitales.

Esta aplicación ha sido desarrollada para su utilización junto con la placa de pruebas desarrollada y descrita en el apartado 4.2. En ella se disponían una serie de leds en el puerto digital D de entrada/salida y de tres potenciómetros conectados a los canales de entrada analógica 0, 3 y 5. Éstos potenciómetros de 22k varían las entradas analógicas entre 0 y 5V. La aplicación que se detalla a continuación mide los canales de entrada analógicos ADCIN0 y ADCIN3, y muestra el resultado en los leds del puerto digital D mediante parpadeo de un número determinado de leds. Para realizar esto se emplea el temporizador de uso general GPT1 que genera un periodo de 0.2 segundos (5 Hz) al final del cual arranca la conversión del convertidor A/D. Cuando ésta conversión

ha finalizado, el convertidor produce una interrupción cuya rutina de interrupción "ADC_ISR" se encarga de leer los dos canales y guardar los resultados en dos variables denominadas ADC0_result y ADC3_result. Ambos resultados serán mostrados después de forma alternativa a través de los leds del puerto digital D, es decir, primero se muestra el resultado del canal ADCIN0 y después de un tiempo de retardo se muestra el resultado de ADCIN3, para después volver a mostrar la misma secuencia. Según el valor de los resultados serán mostrados un número mayor o menor de leds encendidos que parpadearán debido a que son presentados de forma alternativa ambos resultados. El número de leds encendidos será calculado realizando varias operaciones con los bits de los resultados obtenidos. Al leer los registros FIFO el resultado se localizaba en los 10 bits más significativos del registro, como se detalló anteriormente. Para obtener el valor del resultado se desplazan los bits 6 posiciones a la derecha, con lo que éstos variarán entre 0 y 1023. El número de leds que serán encendidos se calculan comprobando el valor de los 4 bits más significativos del resultado obtenido, dando como resultado los siguientes casos:

ADCx_result	Leds On	Rango de ADCx_result	Rango de Tensiones
0000 0000 00	Ninguno	0 – 64	0 – 0.31 V
0001 xxxx xx	0	64 – 128	0.31 – 0.63 V
001x xxxx xx	0 y 1	128 – 256	0.63 – 1.25 V
010x xxxx xx	0 a 2	256 – 384	1.25 – 1.87 V
011x xxxx xx	0 a 3	384 – 512	1.87 – 2.5 V
100x xxxx xx	0 a 4	512 – 640	2.5 – 3.12 V
101x xxxx xx	0 a 5	640 – 768	3.12 – 3.75 V
110x xxxx xx	0 a 6	768 – 896	3.75 – 4.38 V
111x xxxx xx	0 a 7	896 – 1023	4.38 – 5 V

En la tabla anterior se muestran el número de leds encendidos según el rango de valores en el que se encuentre el resultado obtenido, lo que equivale a un rango de tensiones en la entrada analógica. De ésta tabla se desprende que cada caso es identificado por los valores de los 4 bits más significativos del resultado (resaltados en negrita): los dos primeros casos los bits valen 0 y 1, y en el resto se corresponden con el 1, 2, 3, 4, 5, 6 y 7. A partir de esto, la programación de la aplicación va a identificar a cada caso con el número al que equivalen, por lo que va ha realizar dos operaciones más de desplazamiento:

- una de 6 bits a la derecha para quedarse con los 4 bits más significativos, con los que se identificaran los dos primeros casos (0 y 1).

- y otro desplazamiento de un bit para quedarse con los 3 bits más significativos, con los que identificar el resto de los casos mediante el valor de dichos bits (de 1 a 7).

El resultado de estos desplazamientos los guardará en una variable global denominada "result". Para diferenciar cada uno de los casos va a emplear la directiva "switch(result)", la cual elige entre los distintos casos que se han detallado según el valor de "result". Todo este algoritmo se representa en el código mediante las siguientes líneas:

```
result>>=6;
switch(result) {
     case 0 : PDDATDIR=0xFF00;break;
     case 1 : PDDATDIR=0xFF01;break;
     }
result>>=1;
switch(result) {
     case 1 : PDDATDIR=0xFF03;break;
     case 2 : PDDATDIR=0xFF07;break;
     case 3 : PDDATDIR=0xFF0F;break;
     case 4 : PDDATDIR=0xFF1F;break;
     case 5 : PDDATDIR=0xFF3F;break;
     case 6 : PDDATDIR=0xFF7F;break;
     case 7 : PDDATDIR=0xFFFF;break;
     }
```

De esta forma la opción "switch(0)" emplearía el caso 0, con lo que no se enciende ningún led, o la opción "switch(5)" que elegiría el caso 5 encendiendo 6 leds (FF3F = 1111 1111 0011 1111 → enciende del 0 al 5). Todo este algoritmo lo realiza la función "show_ADC" que se encarga de mostrar el resultado de la medida en los leds del puerto digital D.

Los archivos necesarios para realizar este proyecto son, básicamente, los mismos que los utilizados en ejemplos anteriores. Únicamente varían el archivo con el programa principal y el archivo de vectores de interrupción, por los que serán los únicos detallados a continuación.

- ***f243adc2.c:*** Fichero principal que contiene el código en C para este ejemplo.
- ***vectors.asm:*** Fichero donde se definen los saltos a las distintas interrupciones.
- ***regs243.h:*** Archivo include donde se definen los registros de control mapeados en memoria de los distintos periféricos.
- ***F243adc.cmd:*** Fichero empleado por el enlazador (linker) para la ubicación de variables y constantes en la memoria del DSP (idéntico a los casos anteriores).
- ***rts2xx.lib:*** Librería de funciones empleadas en C por Code Composer.
- ***wait.asm:*** Archivo que contiene la función de configuración del generador de estados de espera.

- **Fichero f243adc2.c:**

```
/*****************************************************************/
/* Fichero: f243adc2.c                                           */
/* Programa para test del convertidor ADC                        */
/* para TMS320F243, reloj externo de 5MHz, interno de 20Mhz      */
/*****************************************************************/
/* Dos potenciómetros ( 0 a 5V) conectados en canales ADCIN0 y ADCIN3 */
/* 8 LED's conectados en puerto D0...D7 ; LED-on : 1  LED off : 0 */
/* Timer GPT1 genera periodo de 0.2 segundos (f = 5 Hz)          */
```

```
/* Cuando se alcanza el periodo se inicia la conversión de las dos entradas analógicas */
/* Convertidor ADC1: ADCIN0 y Convertidor ADC2: ADCIN3                                 */
/* La rutina de interrupción ADC_ISR lee los resultados de las conversiones            */
/* La función show_ADC muestra los resultados en los leds del puerto D                 */
/*************************************************************************/

#include "regs243.h"   /* Inclusión del archivo de definición de etiquetas de registros */

/*********************   SETUP del registro OCRA   **********************/
#define OCRA15                  0       /* 0 : IOPB7   1 : TCLKIN              */
#define OCRA14                  0       /* 0 : IOPB6   1 : TDIR                */
#define OCRA13                  0       /* 0 : IOPB5   1 : T2PWM               */
#define OCRA12                  0       /* 0 : IOPB4   1 : T1PWM               */
#define OCRA11                  0       /* 0 : IOPB3   1 : PWM6                */
#define OCRA10                  0       /* 0 : IOPB2   1 : PWM5                */
#define OCRA9                   0       /* 0 : IOPB1   1 : PWM4                */
#define OCRA8                   0       /* 0 : IOPB0   1 : PWM3                */
#define OCRA7                   0       /* 0 : IOPA7   1 : PWM2                */
#define OCRA6                   0       /* 0 : IOPA6   1 : PWM1                */
#define OCRA5                   0       /* 0 : IOPA5   1 : CAP3                */
#define OCRA4                   0       /* 0 : IOPA4   1 :CAP2/QEP2            */
#define OCRA3                   0       /* 0 : IOPA3   1 : CAP1/QEP1           */
#define OCRA2                   0       /* 0 : IOPA2   1 :XINT1                */
#define OCRA1                   0       /* 0 : IOPA1   1 :SCIRXD               */
#define OCRA0                   0       /* 0 : IOPA0   1 : SCITXD              */
/*************************************************************************/
/*********************   SETUP del registro OCRB   **********************/
#define OCRB9                   0       /* 0 : IOPD1   1 : XINT2/EXTSOC        */
#define OCRB8                   1       /* 0 : CKLKOUT  1 : IOPD0              */
#define OCRB7                   0       /* 0 : IOPC7   1 : CANRX               */
#define OCRB6                   0       /* 0 : IOPC6   1 : CANTX               */
#define OCRB5                   0       /* 0 : IOPC5   1 : SPISTE              */
#define OCRB4                   0       /* 0 : IOPC4   1 : SPICLK              */
#define OCRB3                   0       /* 0 : IOPC3   1 : SPISOMI             */
#define OCRB2                   0       /* 0 : IOPC2   1 : SPISIMO             */
#define OCRB1                   1       /* 0 : BIO     1 : IOPC1               */
#define OCRB0                   1       /* 0 : XF      1 : IOPC0               */
/*************************************************************************/
/*********************   SETUP del WDCR   ********************************/
#define WDDIS                   1       /* 0 : Watchdog enabled 1: disabled    */
#define WDCHK2                  1       /* 0 : System reset   1: Normal OP     */
#define WDCHK1                  0       /* 0 : Normal Oper.     1: sys reset   */
#define WDCHK0                  1       /* 0 : System reset   1: Normal OP     */
#define WDSP                    7       /* Watchdog prescaler 7 : div 64       */
/*************************************************************************/
/*********************   SETUP del registro SCSR   **********************/
#define CLKSRC                  0       /* 0 : interno(20MHz)                  */
#define LPM                     0       /* 0 : Modo bajo consumo 0 si IDLE     */
#define ILLADR                  1       /* 1 : borrar ILLADR                   */
/*************************************************************************/
```

```
/************************    SETUP de WSGR    ************************/
#define BVIS                0       /* 10-9 : 00, Bus visibility OFF                */
#define ISWS                0       /* 8 -6 : 000, 0 estados de espera para I/O     */
#define DSWS                0       /* 5 -3 : 000, 0 estados de espera para  datos  */
#define PSWS                0       /* 2 -0 : 000, 0 estados de espera para código  */
/*********************************************************************/
/********************     SETUP del registro GPTCON    *****************/
#define GPTCON_T2TOADC      0
                                    /* 10-9  : T2TOADC = 00 : GPT2 no inicia ADC    */
#define GPTCON_T1TOADC      2
                                /*  8-7  : T1TOADC = 10 : GPT1  inicia ADC por periodo */
#define GPTCON_TCOMPOE      0
        /*   6   : TCOMPOE = 0  : deshabilitadas las 2 salidas de comparación del GPT */
#define GPTCON_T2PIN        0
                            /*  3-2  : T2PIN = 00 : Pol. de salida GPT2 = forzada nivel bajo */
#define GPTCON_T1PIN        0
                            /*  1-0  : T1PIN = 00 : Pol. de salida GPT1 = forzada nivel bajo */
/*********************************************************************/
/********************     SETUP del registro T1CON    *****************/
#define T1CON_FREESOFT      0
                        /* 15-14 FREE, SOFT : 00 stop en emulación JTAG suspendido   */
#define T1CON_TMODE         2
                            /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T1CON_TPS           7
                    /* 10-8  : TPS2-0     : 111  prescaler del reloj de entrada CPUCLK/128  */
#define T1CON_TENABLE       1       /*  6    : TENABLE    : 1 habilita GPT1      */
#define T1CON_TCLKS         0
                                    /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T1CON_TCLD          1
                            /* 3-2   : TCLD1,0    : 01 Recarga del timer 1 al llegar a 0 o a T */
#define T1CON_TECMPR        0
                    /* 1     :  TECMPR     : 0 deshabilita operación de comparación del timer */
/*********************************************************************/
/********************    SETUP del registro EVIMRA    *****************/
#define T1OFINT             0  /* 10 : Timer 1 overflow interrupt              */
#define T1UFINT             0  /*  9 : Timer 1 underflow interrupt             */
#define T1CINT              0  /*  8 : Timer 1 compare interrupt               */
#define T1PINT              0  /*  7 : Timer 1 period interrupt                */
#defineCMP3INT              0  /*  3 : Compare 3 interrupt                     */
#define CMP2INT             0  /*  2 : Compare 2 interrupt                     */
#define CMP1INT             0  /*  1 : Compare 1 interrupt                     */
#define PDPINT              0  /*  0 : Power Drive Protect Interrupt           */
/*********************************************************************/
/********************    SETUP del registro EVIMRB    *****************/
#define T2OFINT             0  /*  3 : Timer 2 overflow interrupt              */
#define T2UFINT             0  /*  2 : Timer 2 underflow interrupt             */
#define T2CINT              0  /*  1 : Timer 2 compare interrupt               */
#define T2PINT              0  /*  0 : Timer 2 period interrupt                */
/*********************************************************************/
```

```
/********************    SETUP del registro EVIMRC    *********************/
#define CAP3INT                 0  /*  2 : Capture Unit 3 interrupt                */
#define CAP2INT                 0  /*  1 : Capture Unit 2 Interrupt                */
#define CAP1INT                 0  /*  0 : Capture unit 1 interrupt                */
/*****************************************************************************/
/***********************    SETUP del registro IMR    ************************/
#define INT6                  1  /*  5 : Level INT6  no enmascarada                */
#define INT5                  0  /*  4 : Level INT5  enmascarada                   */
#define INT4                  0  /*  3 : Level INT4  enmascarada                   */
#define INT3                  0  /*  2 : Level INT3  enmascarada                   */
#define INT2                  0  /*  1 : Level INT2  enmascarada                   */
#define INT1                  0  /*  0 : Level INT1  enmascarada                   */
/*****************************************************************************/
/********************  SETUP del registro ADCCTRL1   *********************/
#define SOFTFREE            2  /* 15-14 : 10 completa conversión antes de parar      */
#define ADCIMSTART      0  /* 13 :  no comienza conversión inmediatamente      */
#define ADC2EN              1  /* 12 :  unidad ADC2 habilitada                     */
#define ADC1EN              1  /* 11 :  unidad ADC1 habilitada                     */
#define ADCCONRUN       0  /* 10 :  conversión en modo no continuo             */
#define ADCINTEN            1  /* 9   :  habilita interrupción del ADC             */
#define ADCINTFLAG      1  /* 8   :  borra interrupciones previas              */
#define ADC2CHSEL       3  /* 6-4:  011 Canal 3 para unidad ADC2               */
#define ADC1CHSEL       0  /* 3-1:  000 Canal 0 para unidad ADC1               */
#define ADCSOC              1  /* 0   :  Inicio de conversión                      */
/*****************************************************************************/
/********************  SETUP del registro ADCCTRL2   *********************/
#define ADCIM               1  /* 14 : flag de interrupción del ADC                */
#define ADCINTPRI       1  /* 11 : prioridad de interrupción de ADC en nivel 6 */
#define ADCEVSOC        1  /* 10: habilitado inicio ADC por Gestor de Eventos  */
#define ADCEXTSOC       0  /*  9 : deshabilita inicio externo del ADC          */
#define ADCPSCALE       7  /* 2-0: 000 prescaler CLKOUT/1                      */
/*****************************************************************************/

#define PERIOD 31250        /* periodo de Timer 1 = 50ns * 128 * 31250 = 0.2s */

unsigned int ADC0_result,ADC3_result;              /* declara variables resultado */

interrupt void ADC_ISR(void);                      /* declara rutina de interrupción */

void show_ADC(unsigned int result);         /* declara función muestra de resultados */

void c_dummy1(void);                       /* declaración función c_dummy */

extern _out_wsgr();             /* declara función de configuración de estados de espera */

void c_dummy1(void)
{
        while(1);               /* función para atrapar interrupciones espúreas */
}
```

```
/******************  Función para muestra de resultados  ******************/
/* Función que muestra el resultado de las conversiones en los leds del puerto D      */
/* Los resultados son almacenados en ADCx_result por la rutina ADC-ISR                */
/* Se encienden tantos leds según el análisis del valor de los 4 bits más significativos */
/* 0000 0000 00  =  todos los LED'S off – Caso 0                                      */
/* 0001 xxxx xx  =  LED 0 on – Caso 1                                                 */
/* 001x xxxx xx  =  LED´s 0 y 1 on – Caso 1                                           */
/* 010x xxxx xx  =  LED´s 0, 1 y 2 on – Caso 2                                        */
/* 011x xxxx xx  =  LED´s 0, 1, 2 y 3 on – Caso 3                                     */
/* 100x xxxx xx  =  LED´s 0, 1, 2, 3 y 4 on – Caso 4                                  */
/* 101x xxxx xx  =  LED´s 0, 1, 2, 3, 4 y 5 on – Caso 5                               */
/* 110x xxxx xx  =  LED´s 0, 1, 2, 3, 4, 5 y 6 on – Caso 5                            */
/* 111x xxxx xx  =  todos los LED´s encendidos – Caso 5                               */
/*********************************************************************************/

void show_ADC(unsigned int result)
{
        result>>=6;                     /* desplaza resultado 6 bits a la derecha */
        switch(result) {                                /* selección del estado a mostrar */
                case 0 : PDDATDIR=0xFF00;break;  /* analizando valor de los 4 bits */
                case 1 : PDDATDIR=0xFF01;break;          /* más significativos */
                }

        result>>=1;                     /* desplaza resultado 1 bits a la derecha */
        switch(result) {                                /* selección del estado a mostrar */
                case 1 : PDDATDIR=0xFF03;break;   /* analizando valor de los 3 bits */
                case 2 : PDDATDIR=0xFF07;break;          /* más significativos */
                case 3 : PDDATDIR=0xFF0F;break;
                case 4 : PDDATDIR=0xFF1F;break;
                case 5 : PDDATDIR=0xFF3F;break;
                case 6 : PDDATDIR=0xFF7F;break;
                case 7 : PDDATDIR=0xFFFF;break;
                }
}

/*************  Rutina de tratamiento de la interrupción del ADC  **************/

interrupt void ADC_ISR(void)
{
 if((PIVR-0x0004)==0)       /*Verifica origen de interrupción ( 4 = ADC ) */
        {
        ADC0_result=ADCFIFO1>>6;        /* extrae resultados de FIFO desplazando */
        ADC1_result=ADCFIFO2>>6;        /* 6 bits y almacena en ADCx_result */
        ADCTRL1 |= 0x0100;              /* borra flag de interrupción del ADC */
        }
}
```

```
/**********************  Función principal MAIN  **********************/

void main(void)
{
asm (" setc INTM");          /* Deshabilita todas las interrupciones */
asm (" clrc SXM");           /* Borra el bit de extensión de signo */
asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                          /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);        /* Inicializa SCSR     */

out_wsgr((BVIS<<9)+(ISWS<<6)+(DSWS<<3)+PSWS);
                                    /* Función externa para configurar WSGR */

 OCRB = ((OCRB9<<9)+(OCRB8<<8)+
              (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
              (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                  /* Inicializa OCRB */

 OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                  /* Inicializa OCRA */

GPTCON=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
          (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                                  /* Inicia el registro de control del temporizador */

PDDATDIR = 0xFF00;           /* Apaga los leds del puerto D */

T1PR = PERIOD;               /* Configura el periodo del temporizador */
T1CNT= 0x0000;               /* Valor inicial del contador */

 T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+(T1CON_TPS<<8)+
             (T1CON_TCLKS<<4)+(T1CON_TCLD<<2)+(T1CON_TECMPR<<1));
                                  /* Configura el registro de control del temporizador 1 */

ADCTRL1= ((SOFTFREE<<14)+(ADCIMSTART<<13)+(ADC2EN<<12)+
                    (ADC1EN<<11) +(ADCCONRUN<<10) +(ADCINTEN<<9)+
                    (ADCINTFLAG<<8)+(ADC2CHSEL<<4)+(ADC1CHSEL<<1));
                                  /*  Configura registro de control 1 del ADC */

 ADCTRL2= ((ADCIM<<14)+(ADCINTPRI<<11)+
       (ADCEVSOC<<10)+(ADCEXTSOC<<9)+ADCPSCALE);
                                  /*  Configura registro de control 2 del ADC */
```

```
EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
                    (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                                        /* Registro de máscaras del grupo A del EV  */

EVIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                                        /* Registro de máscaras del grupo B del EV  */

EVIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                                        /* Registro de máscaras del grupo C del EV  */

EVIFRA=0xFFFF;      /* Registro de flags del grupo A del EV  */
EVIFRB=0xFFFF;      /* Registro de flags del grupo B del EV  */
EVIFRC=0xFFFF;      /* Registro de flags del grupo C del EV  */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                                                /* Registro de máscaras de interrupción */

IFR=0xFFFF;             /* Borra todas las interrupciones pendientes */

asm (" clrc INTM");             /* Habilita todas las interrupciones */

T1CON=T1CON+(T1CON_TENABLE<<6);                 /* Habilita el GPT1*/

while(1){                                               /* bucle sin fin */
                unsigned int i;
/* show_ADC() muestra los últimos resultados de las conversiones en los LED's, con
un retardo entre resultado de un canal y el otro */

      for(i=0;i<65000;i++)show_ADC(ADC0_result);        /* Resultado de ADCIN0 */
      for(i=0;i<65000;i++)show_ADC(ADC3_result);        /* Resultado de ADCIN3 */

                }
}
```

La función “show_ADC” enciende el número de leds adecuado a cada caso descrito anteriormente dependiendo del resultado de la conversión. En el bucle sin fin se muestran continuamente el resultado de ambos canales, pero ello se hace añadiendo un tiempo de espera entre el resultado de un canal y el otro. Esto se consigue mediante un bucle “for” previo a la muestra del resultado, en el cual se decrementa la constante 65000 hasta llegar a cero, momento en el que se muestra el resultado correspondiente. De esta manera, si uno de los potenciómetros está situado a 0V mientras se modifica el otro, el resultado aparecerá como un conjunto de leds parpadeando, cuyo número aumente o disminuye al variar la tensión con el potenciómetro. Al mostrarse ambos resultados de forma alternativa, si ahora se modifica el potenciómetro puesto inicialmente a 0V, los leds comienzan a quedarse fijos a medida que aumenta el valor de la tensión en ese canal, dejando de parpadear todos los leds que se encuentren encendidos cuando el valor de la tensión en ambos canales sea aproximadamente el mismo.

4.7.4.2. Ejemplos con VisSim:

Para emplear las entradas analógicas del procesador TMS320F243 en el entorno de desarrollo VisSim, basta con seleccionar la opción "digital/Analog input for C24x" (figura 4.74) del menú "Vissim/DSP" de la barra superior de menús del programa (figura 4.54). La opción "digital/Analog output for C24x" no puede emplearse como salida analógica para este procesador, ya que carece de convertidor Digital-Analógico, pero sí como salidas digitales, puesto que ambos bloques son compartidos para señales digitales y analógicas. Pulsando dos veces sobre el bloque añadido se abre la ventana de configuración, en la que se selecciona el tipo de canal como analógico en la casilla "Type", además de indicar el canal que se va a muestrear en la casilla "Channel" (figura 4.75).

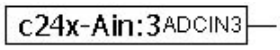


Figura 4.74. Bloque de entrada analógica.

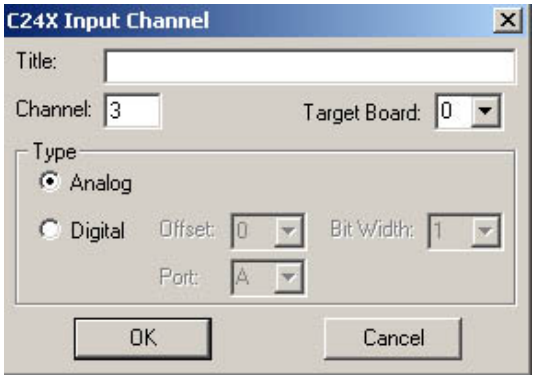


Figura 4.75. Ventana de configuración de entradas analógicas.

De esta forma tan sencilla se tiene seleccionado el canal analógico que se quiere leer. La salida de dicho bloque es el valor de la tensión a la que está sometida dicha entrada analógica, por lo que será un valor entre 0 y 5V.

De igual forma se pueden situar en el diagrama más entradas analógicas con las que realizar cualquier algoritmo. Por ejemplo puede monitorizarse el valor de varias entradas analógicas, para lo cual pueden incluirse displays que muestren el valor de cada uno de los canales analizados. Esto se consigue mediante el icono de la barra superior de menús, el cual permite situar un display y conectarlo a la salida de cualquier bloque. También pueden representarse gráficamente los valores de los canales analizados, para lo cual es necesario situar una gráfica mediante el icono de la barra superior de menús, a la cual se pueden conectar las salidas de los distintos bloques. De esta manera se realiza el diagrama de la figura 4.76, en el cual se muestra el valor de los canales analógicos 0 y 5 en unos displays además de su evolución con el tiempo en una gráfica. Este diagrama puede ejecutarse junto a la placa de pruebas del apartado 4.2, en la que existen dos potenciómetros en los canales 0 y 5 para modificar la tensión en las entradas analógicas citadas. Antes de ejecutar este diagrama deben configurarse las opciones de simulación en la opción "Simulation Properties" del menú "Simulate" del a barra superior de menús. Para su ejecución basta con pulsar el icono "Go" de la barrea de menús.

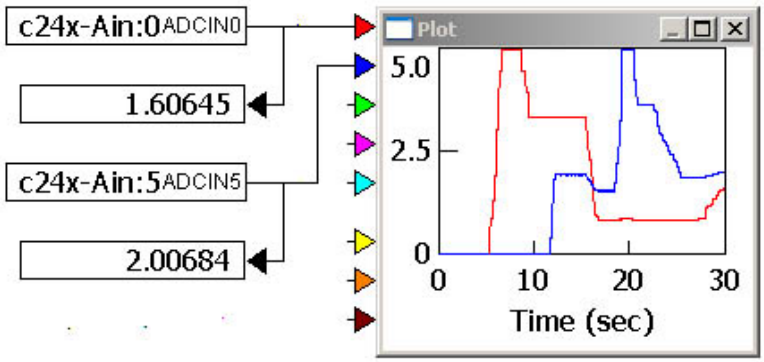


Figura 4.76. Ejemplo de lectura de canales analógicos.

Una vez leído un canal, puede emplearse su valor para introducirlo en un algoritmo. Por ejemplo, puede utilizarse el valor de un canal analógico para controlar el ancho de pulso de una señal PWM. Esto es una modificación del ejemplo de la figura 4.57 del apartado "4.6.8. Periféricos del Gestor de Eventos en VisSim", en el que se mostraba la generación de señales PWM con dicho programa. Su modificación es simplemente sustituir el deslizador que controlaba el ancho de pulso en aquel ejemplo por la entrada analógica. Como la entrada al bloque de generación PWM debe ser un valor entre 0 y 0.99997, debe escalarse el resultado obtenido de la lectura del canal, por lo que se divide dicho valor por la constante 5.0002, como se aprecia en la figura 4.77. De esta forma, la variación de la medida del canal modifica el ancho del pulso generado en lugar de emplear el deslizador.

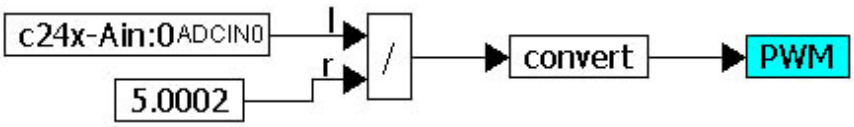


Figura 4.77. Control de señal PWM con lectura analógica.

Este diagrama puede llevarse a cabo también con la placa de pruebas, ya que esta dispone de un potenciómetro en el canal 0 con el que variar la tensión de entrada a dicho canal, de forma que se controle el ancho de la señal PWM. Para comprobar la onda generada puede conectarse un osciloscopio en el pin T1PWM/T1CMP situado en el grupo de pines correspondiente al puerto B de señales digitales. Más concretamente en el pin IOPB4 de dicho puerto.

CAPÍTULO 5: CONTROL DE MOTORES CON DSP

5.1. INTRODUCCIÓN AL CONTROL DE MOTORES:

Habitualmente se han utilizado mucho los motores de continua en la industria por su facilidad de control, por su elevada precisión en la regulación de velocidad y por su elevado par de arranque. Sin embargo, tienen una serie de inconvenientes a tener en cuenta:

- Son caros de mantener puesto que se desgastan mucho con el uso, sobre todo la zona de contacto de las escobillas con el colector, debido a las chispas que se generan entre ambas superficies.
- El rozamiento entre escobillas y colector limita también la velocidad de giro.
- Dan una potencia inferior a los motores de inducción para igual intensidad por fase.
- Son muy voluminosos y pesados.

Actualmente las maquinas de inducción las han sustituido en aquellas aplicaciones en las que se requiere un funcionamiento a velocidad variable. Esto es debido a que tienen:

- Mayor fiabilidad.
- Sencillez constructiva (mayor robustez, menor mantenimiento y menor coste).
- Alta capacidad de sobrecarga.

Por el contrario tienen las siguientes desventajas:

- Mayores pérdidas.
- EL flujo y el par están acoplados (dificultad de control).
- Control sensible a parámetros.
- Mayor rizado de par.
- Electrónica asociada más compleja.

Esto ha hecho que se dé un desarrollo tecnológico considerable para poder controlarlos en velocidad, ya que para ello hay que aplicarles una tensión de frecuencia variable. Antiguamente se solucionó este problema de diversas formas:

- Mediante inserción de resistencias variables en maquinas de rotor bobinado.
- A través de la cascada subsíncrona.
- Rectificando la onda de tensión y ondulándola a la frecuencia deseada mediante electrónica de potencia (diodo de potencia, SCR).

Estos métodos de control daban lugar a montajes analógicos muy complejos y que no permitían asegurar el correcto disparo que se estaba imponiendo ni permitía un cambio sencillo de la estrategia impuesta. Por ello, su uso generalizado en aplicaciones de regulación de velocidad no ha llegado hasta esta última década, debido a la necesidad de disponer de una gran capacidad de computación en el dispositivo, ya que los modernos algoritmos de control requieren multiplicación y acumulación de alto rendimiento. Actualmente se está expandiendo cada vez más su uso gracias al:

- Desarrollo de dispositivos electrónicos de potencia (IGBT) de fácil control, alta frecuencia de conmutación y capacidad media de potencia;
- Desarrollo de sistemas de control digital (DSP) con gran potencia computacional y recursos propios de microcontroladores de gama alta que permiten la implementación en tiempo real de complejos algoritmos de control; y
- Disminución de coste de dichos dispositivos y sistemas.

El convertidor CA/CA indirecto con circuito intermedio de tensión es el más utilizado para el control del motor de inducción (figura 5.1).Consta de un rectificador no controlado alimentado desde la red alterna, una capacidad de alto valor y un inversor trifásico. Se suele incluir también un troceador o chopper que permite el frenado del motor. Es importante escoger una estrategia de conmutación para el inversor que sea de fácil realización y permita aplicar al motor la tensión y frecuencia deseadas. La estrategia **PWM** (modulación del ancho de pulso) es casi un estándar. Con esta estrategia, la tensión de salida del inversor se constituye de intervalos de conducción de cada uno de los estados posibles del inversor.

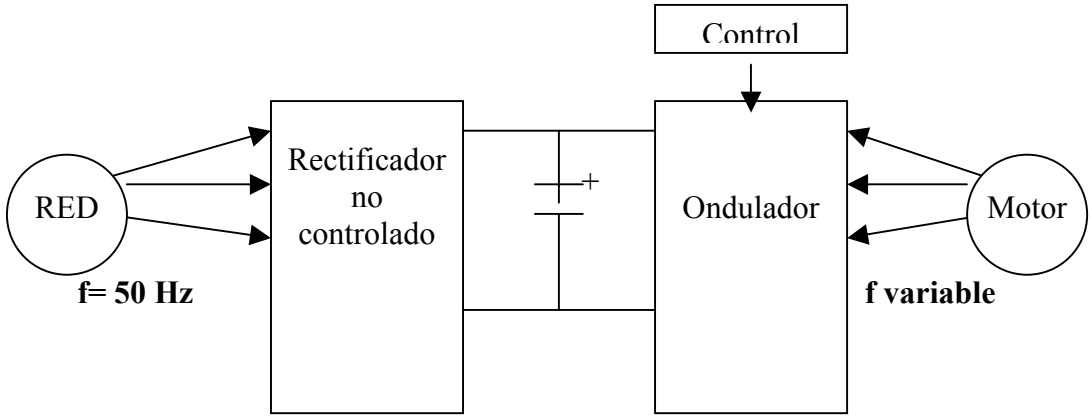


Figura 5.1. Rectificador no controlado y ondulador.

El control para la regulación de velocidad en el motor de inducción es complejo si se desea aprovechar al máximo las prestaciones del motor en cualquier punto de trabajo. La variación de la frecuencia de la tensión de alimentación permite regular la velocidad del motor, pero también provoca una variación indeseada del flujo y del par en el motor, debido al fuerte acoplamiento de las variables. El control escalar, o V/f constante, tiene en cuenta dicho acoplamiento e intenta que el flujo sea constante, para poder suministrar par máximo a cualquier velocidad. El control escalar ofrece una respuesta dinámica lenta e imprecisa, pero es una buena aproximación cuando las exigencias de control no son estrictas. Lo ideal seria conseguir desacoplar las variables del motor de inducción. De esta forma se conseguiría un control independiente de velocidad y par, equiparables a la sencillez en el control del motor de continua, ya que la respuesta dinámica y la precisión mejorarían respecto al control escalar, pero esto implicaría la necesidad de utilizar complicados algoritmos de control. Por otro lado, esto trae consigo una serie de inconvenientes:

- Es necesario conocer el valor de algunos parámetros del motor para el cálculo de los valores deseados de las variables internas. De no conocerse

con exactitud dichos parámetros, aparecen problemas de imprecisión. El problema surge cuando alguno de los parámetros requeridos no es directamente mensurable y debe estimarse (p.e. la resistencia del rotor);

- La gran cantidad de operaciones matemáticas que deben calcularse por muestra para permitir su realización en tiempo real, aunque las frecuencias de muestreo empleadas en el sistema de control de motores no son muy elevadas (algunos kHz) y el flujo de datos de E/S es relativamente bajo.

Existen, de forma general, dos posibles formas de realizar los algoritmos:

- Los algoritmos cableados, a través de FPGA, son la forma más rápida de realizar los cómputos, sin embargo, su falta de versatilidad los circunscriben a soluciones muy particulares en aplicaciones con restricciones temporales muy fuertes.
- La solución programada resulta más interesante dada la posibilidad de reconfigurarla, lo cual comercialmente es mucho más atractivo.

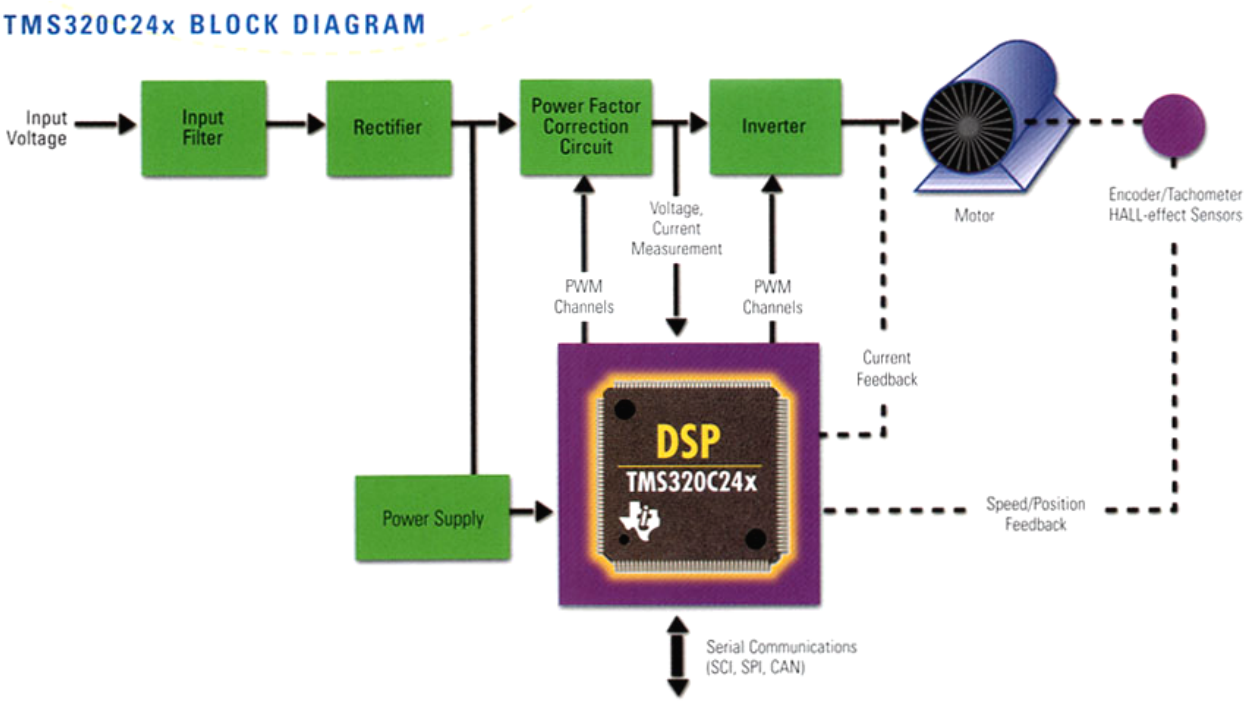


Figura 5.2. Esquema básico de control de un motor con DSP.

Es esta ultima solución (figura 5.2) a la que actualmente se está tendiendo, ya que la aparición de los primeros procesadores DSP al principio de la década de los 80 facilitó el desarrollo de accionamientos para motores y permitió vencer el inconveniente de la gran carga aritmética que lleva asociada su control. Aunque inicialmente su coste no permitiera competir con las técnicas convencionales de control escalar, los nuevos controladores DSP, como el TMS320x24x, permiten en la actualidad dar soluciones compactas y de bajo coste a este tipo de aplicaciones. Estos DSP´s están diseñados para mejorar el rendimiento de sistemas, reducir su coste y disminuir el número de componentes en sistemas digitales de motores y de control de movimiento. Esto es debido a que permiten el accionamiento directo a velocidades variables de motores,

eliminando o reduciendo la necesidad de correas, engranajes, sensores, hidráulica, poleas y contrapesos. También dan lugar a un rendimiento del sistema más robusto gracias al uso de modernos algoritmos de control adaptativo e inteligente, que posibilitan el funcionamiento de un motor a precisamente las velocidades necesarias, adaptando la carga, la temperatura y otros parámetros. Ello mejora la eficacia energética del motor y su fiabilidad, reduciendo el ruido al mejorar la ondulación del par de torsión, y disminuyendo el coste del sistema ya que decrece el numero de piezas y los costes de mantenimiento. Por ello son utilizados en sistemas de control de automoción (dirección electrónica, frenos antibloqueo), válvulas y controles de calefacción, ventilación y aire acondicionado, compresores y bombas de calor, sistemas de automatización de fabricas, aparatos electrodomésticos y productos de oficina. En todas estas aplicaciones se consigue un ahorro de energía al permitir utilizar motores más pequeños y disminuir o eliminar enlaces mecánicos y otros componentes.

Debido a todas estas ventajas que presenta el DSP, se ha seleccionado una tarjeta de iniciación basada en el procesador TMS320F243 para el desarrollo de la aplicación que será desarrollada a lo largo de este capítulo. Este procesador fue de los primeros en aparecer para el control digital de motores y presenta una serie de características que lo hacen especialmente indicado para el comienzo del control de motores con DSP´s. Esto se debe a que es el más sencillo de los procesadores de la serie C2000 que la empresa Texas Instruments ha fabricado específicamente para el control de motores. A lo largo de capítulos anteriores se han desarrollado las distintas cualidades y características que presenta este dispositivo, cuya aplicación al control de motores se detallan extensamente en este capítulo, en particular, para el control de un motor monofásico de corriente continua.

5.2 SEÑALES DE CONTROL PWM:

Como se comentó anteriormente, la generación de señales PWM es muy utilizada en el control de convertidores electrónicos de potencia y control de motores. Una señal con modulación del ancho del pulso (PWM) es una secuencia de pulsos de frecuencia fija y ancho variable en la que existe un solo pulso por periodo. La frecuencia fija de la señal PWM se denomina frecuencia portadora, y el ancho de los pulsos PWM se determina a partir de una señal de control denominada señal moduladora, la cual suele ser de una frecuencia mucho menor que la de la señal PWM.

Existen diversos métodos para la generación de señales PWM según sea el objetivo que se persiga con la misma. Por ejemplo, en el caso de una modulación PWM senoidal (SPWM), se desea obtener a la salida del convertidor una señal senoidal de amplitud y frecuencia fijas. Para conseguirlo se compara una señal de control senoidal de baja frecuencia (moduladora)con una señal triangular de alta frecuencia (portadora), la cual determina la frecuencia de conmutación de los interruptores controlados. Por lo general la señal triangular es fija, mientras que la señal de control varía su amplitud y frecuencia dependiendo de las necesidades de la carga. De la comparación de ambas señales se obtienen las señales de disparo de los interruptores controlados (figura 5.3), y que son señales de frecuencia fija y ancho de pulso variables, por tanto PWM.

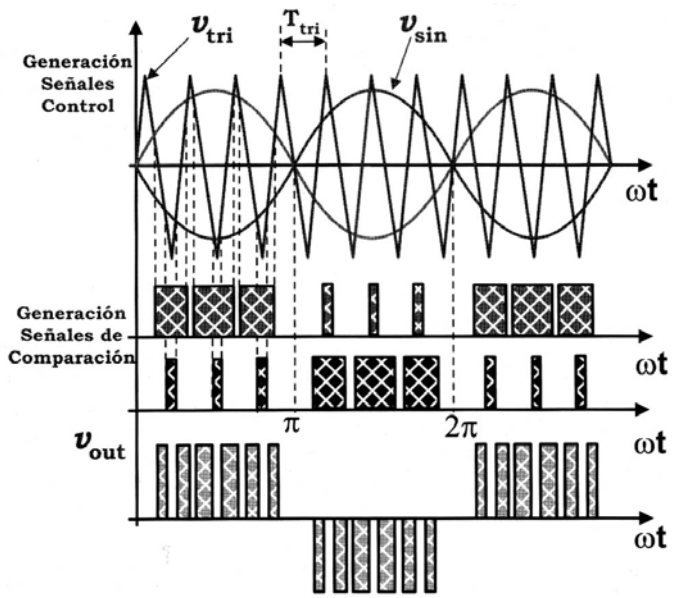


Figura 5.3. Generación analógica de señales PWM.

Habitualmente los convertidores se emplean para el control de motores. En este tipo de sistemas, las señales PWM se usan para controlar el tiempo que van a estar conduciendo los interruptores de potencia controlados, controlándose así la corriente y la energía que es suministrada al motor. De esta manera, la forma y la frecuencia de las corrientes que circulan por cada fase y la tensión aplicada a cada devanado del motor determinan la velocidad y par del mismo.

La generación de señales PWM necesitaba del montaje de un sistema complejo a base de componentes analógicos con los cuales generar y comparar las señales moduladora y portadora. En la actualidad estos montajes están siendo sustituidos por sistemas digitales, ya que permiten realizar la misma operación de forma más precisa y económica debido a la gran potencia de cálculo de los procesadores actuales, además de la reducción de componentes y de espacio que supone su empleo para el usuario. La generación con técnicas digitales requiere de un temporizador que tenga un periodo de conteo igual al periodo PWM deseado. El valor de este contador es constantemente comparado con el de un registro de comparación, el cual contiene los valores que va tomando en cada ciclo la señal moduladora de control. Cuando se produce una comparación y ambos valores coinciden, se efectúa una transición en las señales de salida asociadas. Al terminar el periodo o al producirse una segunda comparación, existe otra transición de las señales de salida al estado inicial. De esta forma se obtienen pulsos de salida en los que su duración es proporcional al registro de comparación, permitiendo modificar el tiempo que cada interruptor de potencia se encuentra conduciendo.

Generalmente los convertidores de potencia emplean una estructura basada en un par de interruptores controlados ubicados entre el terminal positivo y negativo de la tensión de continua. Cada uno de ellos se controla mediante señales complementadas, de manera que sólo conduzca uno de ellos en cada momento. Este montaje se denomina semipuente. Si el convertidor es monofásico, sólo requiere un semipuente controlado, mientras que en los trifásicos se necesitan tres de ellos conectando la carga en los puntos intermedios de cada semipuente (figura 5.4).

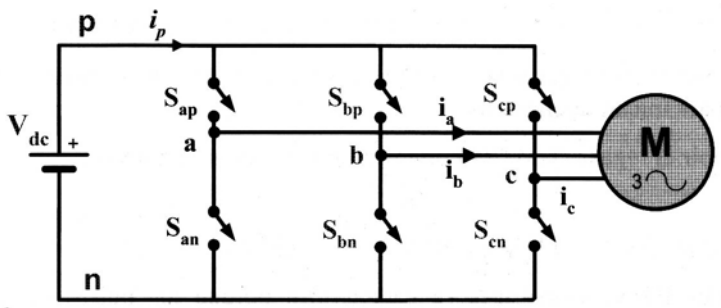


Figura 5.4. Convertidor de potencia trifásico.

Independientemente del tipo de convertidor empleado, en cada semipuente únicamente puede haber un interruptor conduciendo en cada momento, ya que lo contrario produciría un cortocircuito que destruiría los dispositivos. Para prevenir el solapamiento de las señales de control y evitar este problema es necesario la generación de tiempos muertos entre las dos señales de control de un semipuente. Estos tiempos se obtienen a partir de las características de potencia y dependiendo de las características de la carga.

Los procesadores DSP permiten la generación y control de señales PWM, para lo cual disponen de un módulo especial denominado Gestor de Eventos o Event Manager. Este módulo contiene temporizadores, unidades de comparación, unidad de generación de tiempos muertos y una lógica de salida que permite determinar el estado y polaridad de las señales PWM de salida. En particular, el procesador TMS320F243 dispone de tres unidades de comparación con las que generar hasta seis señales PWM, tres de ellas complementadas, de forma que unas controlan la parte superior de cada semipuente y las otras tres la parte inferior. Estas seis salidas, junto a otras dos generadas a partir de dos temporizadores de uso general, pueden emplearse para el control de motores trifásicos o motores DC brushless. La posibilidad de configurar cada salida de forma independiente a las demás, permite también implementar fácilmente el control de motores de reluctancia conmutada, motores DC o motores paso a paso.

El procedimiento para generar varias señales PWM mediante un procesador DSP incluye la configuración de los registros detallados en el capítulo 4, en particular:

- El registro de control de acción de las salidas *ACTR*.
- El registro de control de la unidad de tiempos muertos *DBTCON* si es necesario.
- Iniciar el registro de comparación de cada una de las unidades de comparación *CMPRx*.
- Configurar el registro de control de comparación *COMCON*.
- EL registro contador a emplear para iniciar la operación.
- Ir cambiando el valor del registro comparador *CMPRx* según sea necesario.

Dependiendo de los registros empleados y de la configuración establecida en estos, pueden generarse distintos tipos de señales WPM con el DSP:

- **Señales PWM asimétricas:**
  Este tipo de señales se caracterizan porque los pulsos generados no se encuentran centrados respecto del periodo de la señal (figura 5.5) Por ello, el ancho de cada pulso sólo puede cambiar en uno de los lados del pulso, permaneciendo el otro constante al final o al inicio del periodo.

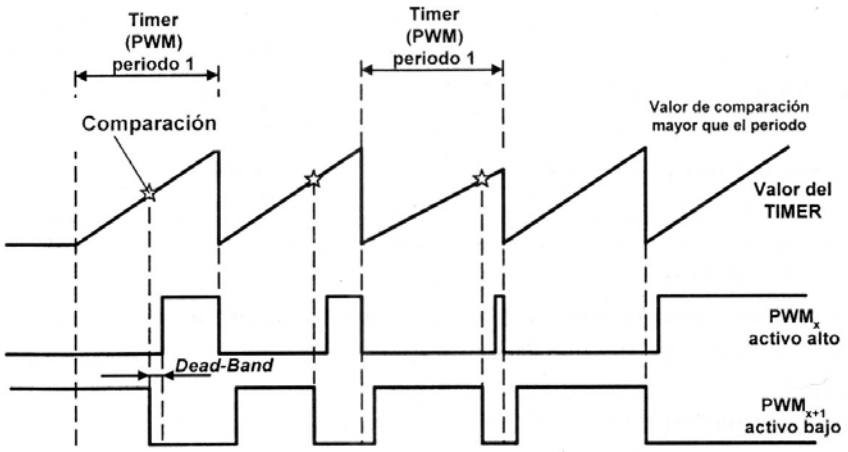


Figura 5.5. Señales PWM asimétricas complementadas.

Como se detalló en el capítulo anterior, las señales PWM asimétricas se generaban empleando el temporizador en modo de cuenta continua ascendente, estableciendo el periodo de la señal PWM en el registro *TxPR*. Por otro lado, en cada periodo deben rescribirse los registros de comparación para modificar el ancho del pulso según los requerimientos de la carga, lo cual se realizaba de forma instantánea al disponer estos registros de otro auxiliar de precarga de los datos.

- **Señales PWM simétricas:**
  Estas señales se caracterizan por estar centrados los pulsos respecto al periodo de la señal PWM (figura 5.6). Esta simetría permite que existan en la señal dos intervalos en estado inactivo de igual duración, uno al principio y otra al final del periodo. Esto produce menos armónicos en las corrientes de fase de los motores trifásicos que las señales asimétricas cuando se implementa una señal PWM senoidal (SPWM).

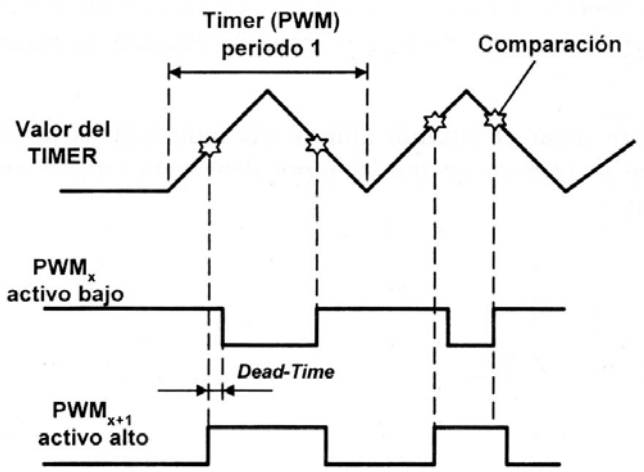


Figura 5.6. Generación de señales PWM simétricas complementadas.

Este modo de generación requería que el temporizador empleado fuese configurado en modo de cuanta continua ascendente/descendente. Ello provoca que existan dos comparaciones, una durante el tramo ascendente y otra en el descendente. El valor de comparación en ambos casos puede ser distinto. Este suele emplearse para modificar la señal PWM y compensar los errores de corrientes debidos a los tiempos muertos introducidos al controlar un motor de alterna.

- **Modulación SVPWM:**
  La modulación PWM en el espacio vectorial (SVPWM) es una técnica aplicada a convertidores de potencia trifásicos. Presenta la ventaja de generar una menor distorsión armónica en las corrientes que circulan por los motores trifásicos además de conseguir una mejor utilización de la tensión de alimentación frente a la modulación PWM senoidal (SPWM).

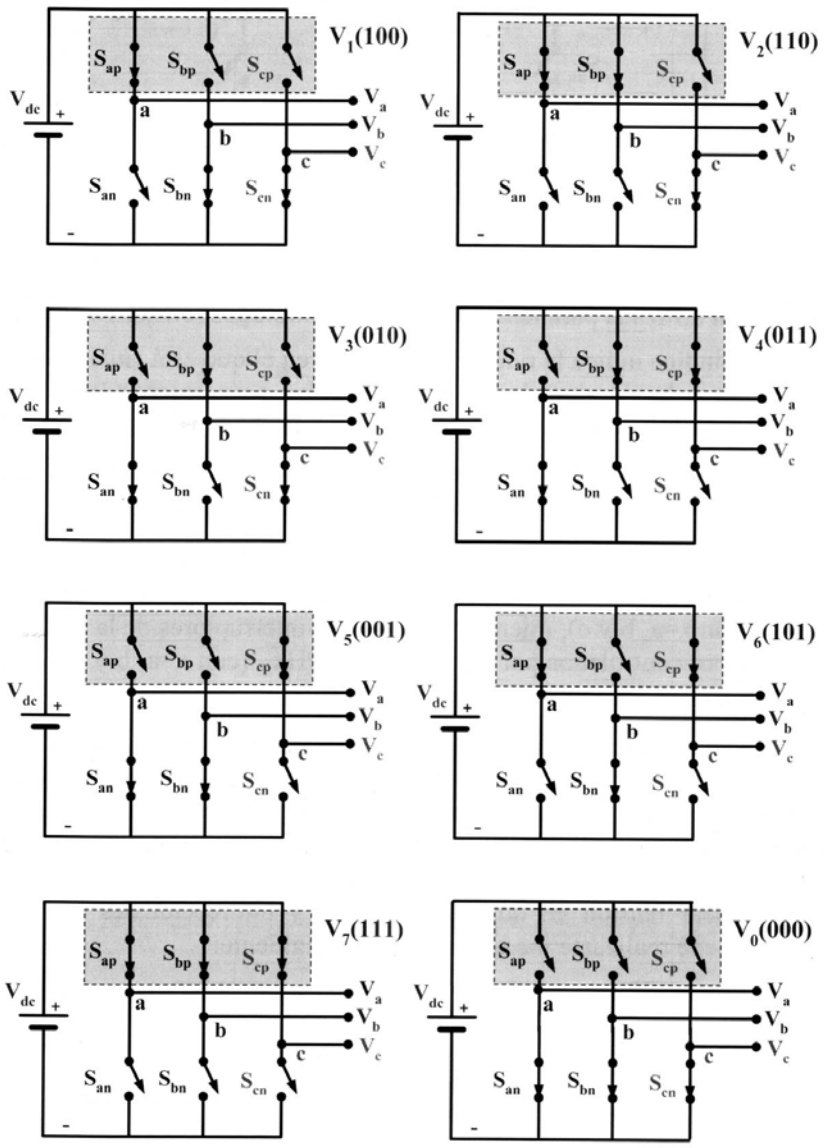


Figura 5.7. Vectores de conmutación en SVPWM.

En un inversor trifásico existen tres semipuentes, en cada uno de los cuales sólo puede haber un interruptor conduciendo, por lo que en una misma rama no pueden dispararse los dos a la vez. Además siempre debe existir al menos uno de ellos conduciendo, de manera que no se dejen en circuito abierto las inductancias de los motores. Como cada interruptor solo puede estar ON y OFF, mediante estas reglas se pueden generar hasta 8 posibles combinaciones de los seis interruptores del convertidor. Si el interruptor conduce se le asigna un 1 y si no conduce un 0. A cada una de estas combinaciones se le denomina "vector de conmutación", que se diferencian uno del anterior en que una de las ramas ha conmutado (figura 5.7). Cada uno de los vectores se corresponde con un valor de tensión determinado entre los terminales de salida de cada una de las fases del convertidor. El objetivo de la modulación SVPWM es conseguir formar el vector que se desea aplicar en el motor como una combinación de los ocho vectores de conmutación. Esto se consigue aplicando el tiempo adecuado y en el orden necesario cada uno de los ocho vectores de conmutación, lo que proporciona la señal de salida del inversor correspondiente. Estos patrones de conmutación del convertidor pueden realizarse definidos por software o por hardware, gracias a las prestaciones que tiene el DSP para ello.

Una vez decidido el tipo de modulación que se desea aplicar a un convertidor, sólo queda diseñar el algoritmo con el que se pretende realizar el control de sus interruptores, para lo que el DSP presenta gran variedad de posibilidades que permiten llegar a soluciones muy efectivas.

5.3 APLICACIÓN PRÁCTICA A UN MOTOR DC:

A continuación, y como aplicación práctica de todo lo desarrollado en capítulos anteriores, se va a desarrollar una sencilla aplicación para el control de la velocidad de un motor de continua. El objetivo es, dado un motor con unas características determinadas, diseñar una placa de control de velocidad basado en el procesador TMS320F243 de Texas Instruments. Dicha placa estará compuesta por una tarjeta de iniciación o Starter Kit basado en dicho procesador y por una placa en la que se situarán el driver que controlará la tensión del motor y de los componentes necesarios para la adaptación de señales que serán medidas por el DSP. El driver de esta placa es un transistor MOSFET de alta frecuencia para ser controlado mediante señales PWM generadas por el DSP. Todo el sistema estará conectado a un PC a través del cual se realizará la programación del algoritmo necesario en el DSP, además de permitir la monitorización de ciertos parámetros del sistema (figura 5.8).

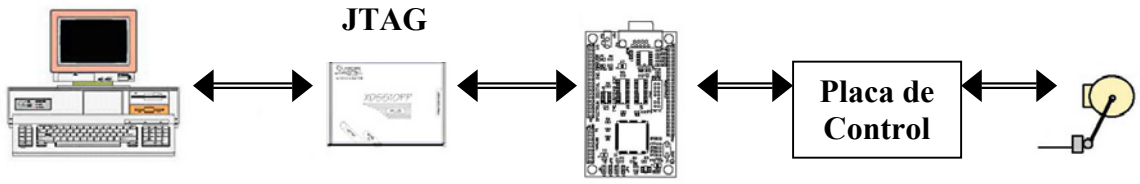


Figura 5.8. Esquema de montaje.

En concreto se desea realizar el control de velocidad de un motor de continua de los disponibles en el laboratorio del departamento de Ingeniería Eléctrica (figura 5.9), el cual dispone de las siguientes características:

- Armadura: alimentación a 24V DC y 5A máximo.
- Campo de excitación: 24V y 2A máximo.
- Potencia de entrada: 125W máximo.
- Potencia de salida: 40W.
- Par: 0.2/0.4 Nm.
- Velocidad: 1000 rpm nominales, 2000 rpm máximo.

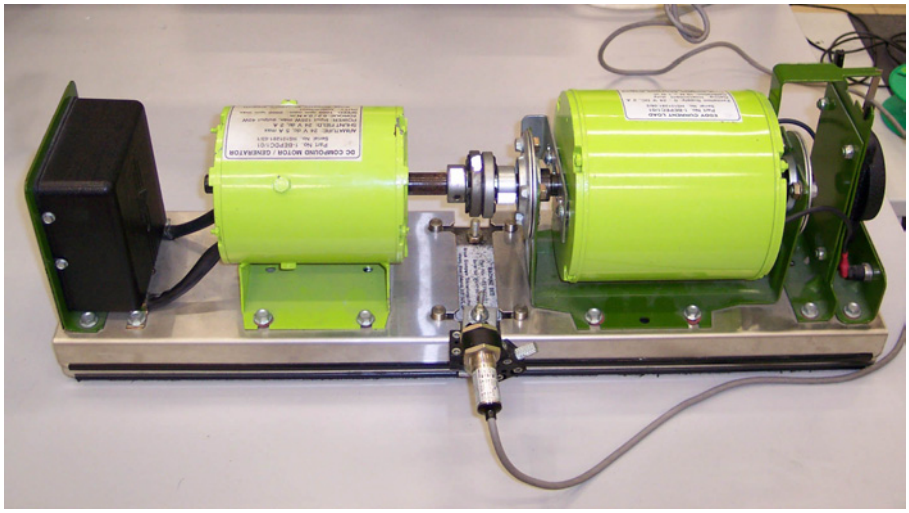


Figura 5.9. Motor empleado con carga.

Este motor permite ser controlado mediante señales PWM de duty cycle variable.

El control de velocidad de un motor DC se realiza variando la tensión media que se le suministra mediante una fuente de tensión: a más tensión aplicada mayor velocidad. Esto se puede realizar recortando la tensión de continua procedente de la fuente de alimentación mediante un transistor, de forma que variando el tiempo de conducción del mismo se aplique al motor una tensión con forma de onda cuadrada, cuyo valor medio depende del ancho que tengan los pulsos en cada ciclo. El montaje necesario para aplicar este tipo de control requiere la conexión del motor en serie con el transistor de control (figura 5.10), de forma que cuando conduzca el transistor el motor es alimentado con la tensión procedente de la fuente de alimentación, y cuando deja de conducir la tensión aplicada es cero. Para evitar la saturación del motor se instala entre las bornas del mismo un diodo de libre circulación, de manera que en los instantes en los que el transistor no conduce la corriente del motor se recircula por dicho diodo.

Para controlar los disparos del transistor se aplicará la técnica de modulación por ancho de pulsos o PWM. Con esta técnica es posible variar el tiempo de conducción del transistor, produciéndose una onda cuadrada de tensión en bornas del motor de valor medio proporcional al ancho de estos pulsos, modificándose así la velocidad del motor cuando sea necesario. Para generar la señal PWM de control del transistor se empleará el procesador TMS320F243 del Starter Kit DSK F243 empleado en esta aplicación.

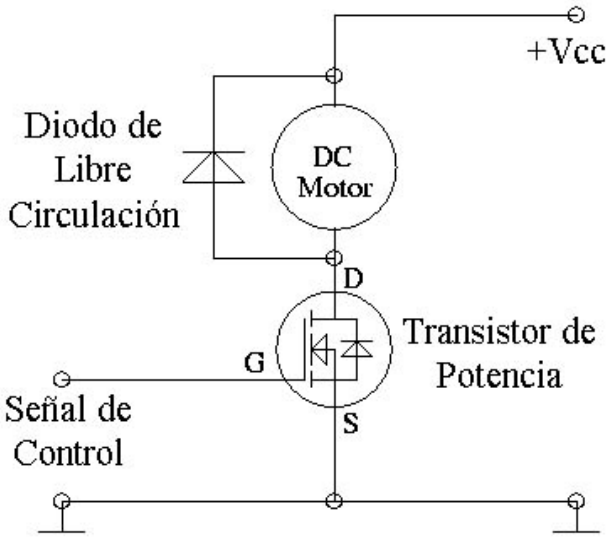


Figura 5.10. Control de un motor DC.

El transistor a emplear en el control de un motor DC es habitualmente un transistor de potencia que, debido a las altas velocidades de conmutación a las que será sometido, suele ser MOSFET. Este tipo de transistores es el más empleado en montajes de control de motores mediante ondas PWM, ya que permite trabajar a altas frecuencias de conmutación y a valores relativamente altos de tensión e intensidad.

5.3.1. Objetivo de la aplicación:

La aplicación que se va a desarrollar pretende controlar la velocidad de un motor de continua mediante el uso de un procesador DSP. En particular se realizará una sencilla aplicación de control a través del valor de tensión de un potenciómetro conectado en una de las entradas analógicas del DSP. De esta forma, mediante el giro del potenciómetro, se introduce un valor de referencia para el DSP con el que se modifica la velocidad del motor DC. Esto se consigue variando el ancho de los pulsos PWM generados por el DSP que controlan el transistor en función del valor de referencia introducido por el potenciómetro en una entrada del convertidor analógico-digital. Con este tipo de control se pretende introducir el uso de un DSP para leer valores de referencia en las entradas del dispositivo con las que actuar en consecuencia para la generación de señales de control.

Por otro lado, la aplicación desarrollada también emplea otros dos canales analógicos del DSP para la medida de la tensión de la fuente de alimentación del motor y de la intensidad que circula por el mismo. Con ambos valores se calcula de forma aproximada la potencia consumida por el motor y se genera una señal PWM de valor medio proporcional a dicha potencia.

Para conseguir realizar las operaciones indicadas, además del transistor de potencia MOSFET, es necesaria la introducción de una serie de elementos que adapten el nivel de tensión de las distintas señales para su lectura o procesamiento por parte del DSP, con lo que el esquema de la figura 5.10 debe ser ampliado para permitir realizar dichas aplicaciones. A continuación se describe el diseño de la placa de control necesaria para la aplicación.

5.3.2. Diseño de la placa de control:

Para poder aplicar un control a un motor DC es necesario realizar un circuito en la que puedan disponerse los distintos componentes que intervienen en el mismo. En este circuito son necesarios dos grupos de componentes:

- **Componentes dedicados al circuito de actuación del motor:**
  Son los dispositivos que se requieren en el sistema para permitir el funcionamiento del motor, como son el transistor de potencia, diodo de libre circulación, tensión de alimentación, condensador para estabilizar la tensión de entrada y diodo de seguridad a la entrada de la alimentación. Estos componentes forman el circuito por el que circula la intensidad del motor, además de encargarse de proporcionar la tensión que alimenta al motor en cada instante (figura 5.11).

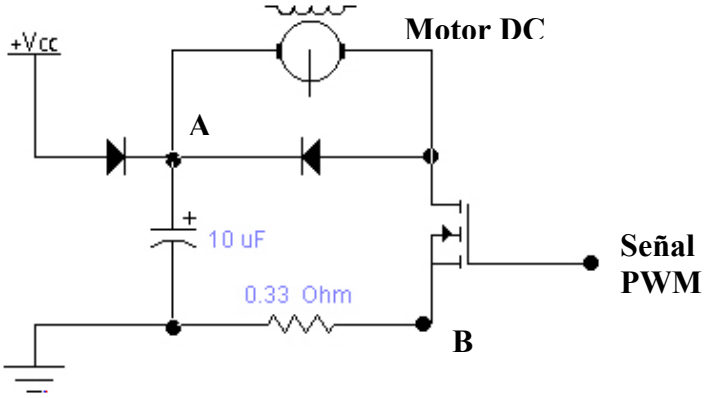


Figura 5.11. Conexiones del motor.

- **Componentes de adaptación de señales de control y de toma de medidas del sistema:**
  Son una serie de componentes adicionales que son necesarios para adaptar los niveles de tensión e intensidad de las señales de control generadas por el DSP. También son necesarios para adaptar los niveles de tensión e intensidad de los puntos donde se realizan medidas para hacerlas compatibles con los niveles requeridos por el convertidor analógico-digital. Estos componentes son resistencias y condensadores para realizar divisores de tensión, filtros, etc. En los puntos A y B de la figura 5.11 es donde se tomarán las medidas que tomará el DSP para el cálculo de la potencia absorbida por el sistema, mientras que a la puerta del transistor llegará la señal de control PWM generada por el DSP.

En el primer grupo de componentes es donde se consume la mayor parte de la potencia del sistema, dado que por ellos circulará una intensidad de valor muy superior al del grupo de medidas. Además están sometidos a la tensión de alimentación. Por ello, los componentes empleados serán escogidos para altas potencias, mientras que los de medida serán componentes de circuitos de pequeña señal comunes en el mercado de la electrónica.

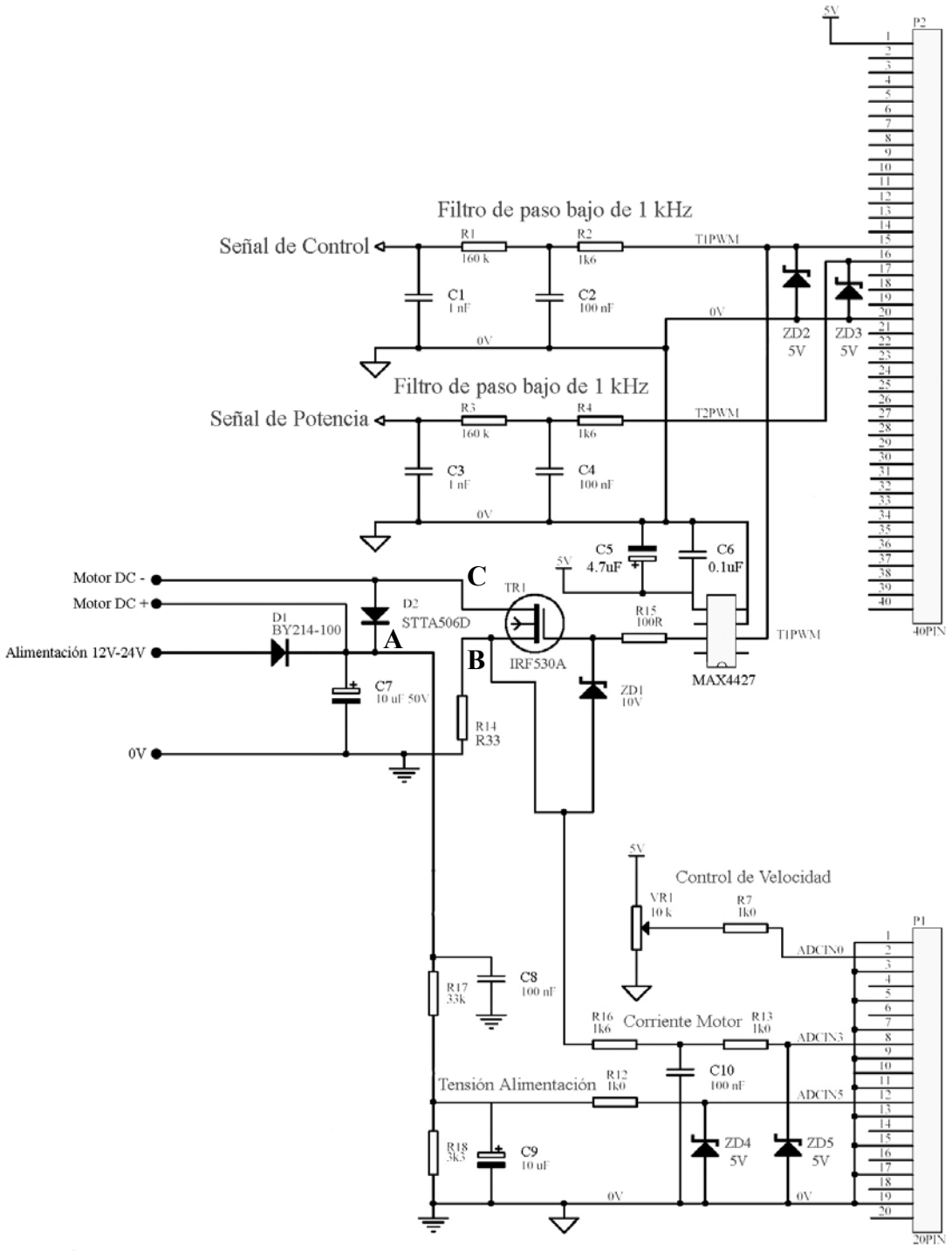


Figura 5.12. Esquema de la placa de control.

En la figura 5.12 se muestra el esquema completo del circuito diseñado. En ella se encuentran los dos grupos de componentes empleados para su construcción. Este esquema es una modificación del mostrado por la figura 5.11, al que se le han añadido los componentes necesarios para adaptar los niveles de las distintas señales, además de incluir elementos de protección para evitar daños al DSP. A continuación se detalla la lista de componentes empleados, así como sus características principales:

- Transistor de potencia MOSFET de 14A y 100V modelo IRF530A.
- Diodo de conmutación rápida de 5A y 600V modelo STTA506D.
- Diodo rectificador de 6A y 100V modelo BY214-100.
- Resistencia de potencia RS de 0.33 Ω y 25W.
- Circuito integrado driver para MOSFET MAXIM 4427.
- 4 diodos de protección Zener de 5V.
- 1 diodo Zener de 10V y 0.5W.
- 1 potenciómetro lineal de 10k multivuelta.
- 2 resistencias de 160k.
- 1 resistencia de 100Ω.
- 3 resistencias de 1k6.
- 3 resistencias de 1k.
- 1 resistencia de 33k.
- 1 resistencia de 3k3.
- 2 condensadores de 1nF.
- 4 condensadores de 100 nF.
- 2 condensadores electrolíticos de 10μF y 50V.
- 1 condensador cerámico de 0.1μF.
- 1 condensador electrolítico de 4.7μF.
- Placa de pruebas taladrada.
- Soportes hexagonales para placa taladrada.
- Radiador 18.205 para diodo rápido.
- Radiador de 3.3ºC/W para transistor y resistencia de potencia.

Algunos de los componentes se han escogido sobredimensionados para ampliar el margen de seguridad, además de permitir emplear una potencia mayor si fuese necesario en el sistema. Los datasheets de los principales componentes se incluyen en el Anexo 4.

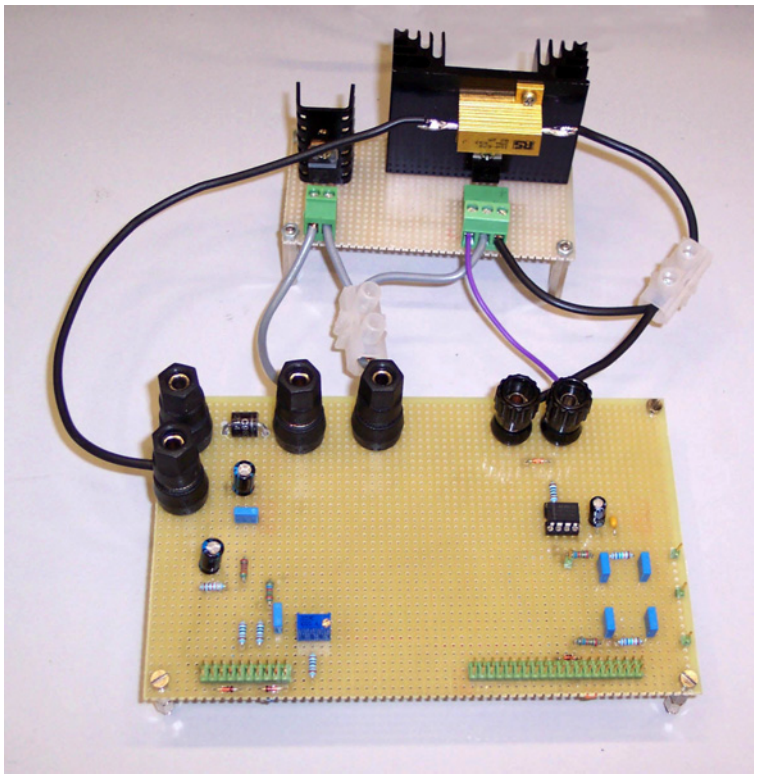


Figura 5.13. Montaje de la placa de control.

A continuación se describen las funciones a las que se dedican algunos componentes y grupos de componentes del circuito:

- **Potenciómetro multivuelta:**
  Se dispone en la placa de un potenciómetro lineal de 10k a través del cual se da una señal de referencia al DSP para generar la señal PWM de control (figura 5.12). Se ha tomado un potenciómetro multivuelta para evitar cambios demasiado bruscos en la velocidad del motor que pudieran provocar picos de intensidad que dañaran algún componente del circuito.

- **Resistencia shunt de 0.33Ω y 25W para medir intensidad del motor:**
  Se ha instalado esta resistencia de pocos ohmios para medir la intensidad que circula por el circuito de potencia de la placa, y por tanto, del motor empleado (figura 5.12). Para realizar las medidas el DSP emplea una entrada al convertidor analógico-digital, el cual sólo mide tensiones en el rango de 0 a 5V. Por ello se emplea la resistencia shunt para medir la tensión del punto B (figura 5.11), cuyo valor está directamente relacionado con el valor de la intensidad que circula por el motor:

  $$V_B = R \cdot I_{motor} = 0.33 \cdot I_{motor}$$

  Se ha calculado la resistencia shunt de tal forma que se alcancen los 5V en el punto B cuando la intensidad que circule sea de 15A, valor muy por encima de lo que llegará a circular por el motor, evitando así dañar la entrada analógica del DSP. Como por el motor pueden circular como máximo 5A, la tensión máxima que medirá el canal 3 del convertidor analógico será:

  $$V_B = 0.33 \cdot I_{motor} = 0.33 \cdot 5A = 1.65A$$

- **Divisor de tensión para medir tensión de alimentación:**
  En el punto A de la figura 5.11 se dispone la tensión suministrada por la fuente de alimentación al circuito. Esta tensión va a ser medida por el canal 5 del convertidor analógico-digital. Es necesario adaptar esta tensión al rango de 0 a 5V que soporta el convertidor. Para ello se dispone un divisor de tensión mediante dos resistencias de 33k y 3k3 (figura 5.12), de manera que para que se alcancen los 5V en el canal 5 de entrada, la tensión suministrada debe ser de:

  $$V_A = \frac{(33 + 3.3)}{3.3} \cdot 5V = 55V$$

  Como la tensión de alimentación que va a ser aplicada es de 24V, el máximo valor leído por el canal 5 será de $V_A = 2.18V$.

- **Diodos Zener de 5V:**
  Se han situado diodos Zener de 5V en los canales de entrada del convertidor analógico-digital y en las señales PWM de salida del DSP (figura 5.12). Esto se ha realizado debido a que estas señales sólo pueden tomar valores entre 0 y 5V, protegiéndolas así de posibles sobretensiones que podrían dañar el DSP.

- **Driver de MOSFET MAX4427:**
  Inicialmente la placa de control se diseñó sin este componente ya que, al ser una señal que atacaba a la puerta de un MOSFET, en principio no debía de haber peligro de dañar por sobretensiones la salida de la señal de control del DSP, pero en las primeras pruebas que se efectuaron con el motor de 24V se dañó la señal PWM2 del DSP. Por esto se decidió instalar en la placa componentes que evitaran sobretensiones en las señales procedentes del DSP, como son los diodos Zener de 5V descritos anteriormente. Para aumentar la seguridad se buscó un componente que se encargara de aislar físicamente las señales de salida del DSP del circuito de potencia. Se decidió emplear el circuito operacional MAX4427 (figura 5.14), específico en el control del disparo de transistores de potencia MOSFET, el cual amplifica el nivel de intensidad procedente de la señal PWM del DSP para asegurar un mejor disparo del transistor. Para su instalación es necesario conectar en paralelo a una de sus patillas dos condensadores: uno cerámico de 0.1μF y otro electrolítico de 4.7μF (figura 5.12), además de conectar dos entradas a tierra según datasheet incluido en Anexo 4.

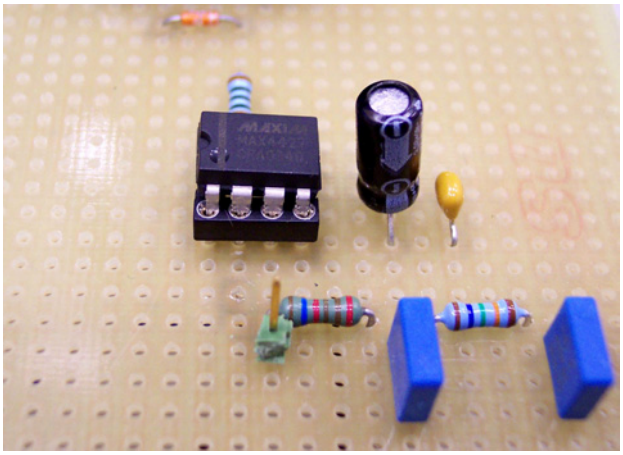


Figura 5.14. Driver de MOSFET MAX4427.

- **Filtro RC en canal analógico de entrada 3:**
  Debido a que la intensidad que circula por el circuito de potencia es intermitente al pasar el transistor de conducción a corte, la tensión del punto B de la resistencia shunt es variable, por lo que para poder medirla correctamente se ha situado un filtro RC a la entrada del canal 3, de manera que sea leído un valor medio de la onda variable de tensión en dicho punto. Este filtro está formado por dos resistencias de 1k y 1k6 y un condensador de 100nF (figura 5.12).

- **Filtros de paso bajo de 1kHz:**
  La aplicación que se ha diseñado genera dos ondas PWM: una dedicada a controlar la tensión aplicada al motor y otra que proporciona una medida de la potencia absorbida por el sistema. El valor medio de ambas señales es proporcional a la magnitud que representan, es decir, el valor medio de la onda de potencia da un valor proporcional a la potencia absorbida, y el valor medio de la señal PWM de control da un valor proporcional a la tensión aplicada al motor. Por ello se disponen en ambas señales filtros de paso-bajo de 1kHz para obtener dicho valor medio, que representado en un osciloscopio proporcionan medidas exactas de ambas magnitudes. Estos filtro se realizan con dos resistencias de 160k y 1k6 además de dos condensadores de 1nF y 100nF (figura 5.12).

El montaje de la placa se ha diseñado de forma que la parte de más potencia del circuito quede separada del resto de componentes dedicados a las medidas. Por esto el circuito ha sido construido en dos placas de prueba independientes:

- **Placa de potencia:**
  En esta parte del circuito se sitúan los componentes que van a ser sometidos a más tensión y por el que va a circular un mayor valor de intensidad. En particular contiene al transistor de potencia, al diodo de libre circulación y la resistencia shunt (figura 5.15). El diodo tiene su propio disipador, mientras que los otros dos componentes comparten un mismo radiador.

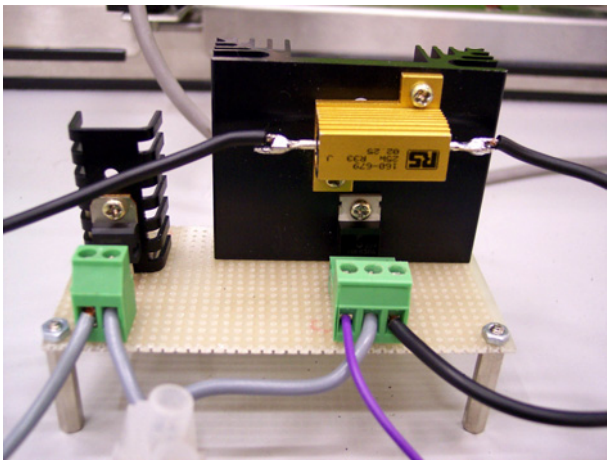


Figura 5.15. Componentes de potencia de la placa de control.

- **Placa de medidas:**
  En ella se disponen los distintos elementos que van a adaptar las señales de control y de medida para hacerlas compatibles con los niveles del DSP. También incluye el diodo y condensador de entrada de tensión de la fuente de alimentación. En esta placa se conectan dicha fuente y el motor a controlar en las bornas de conexión que se han dispuesto para ello (figura 5.16). También se disponen los conectores de 20 y 40 pines para la conexión del Starter Kit F243.

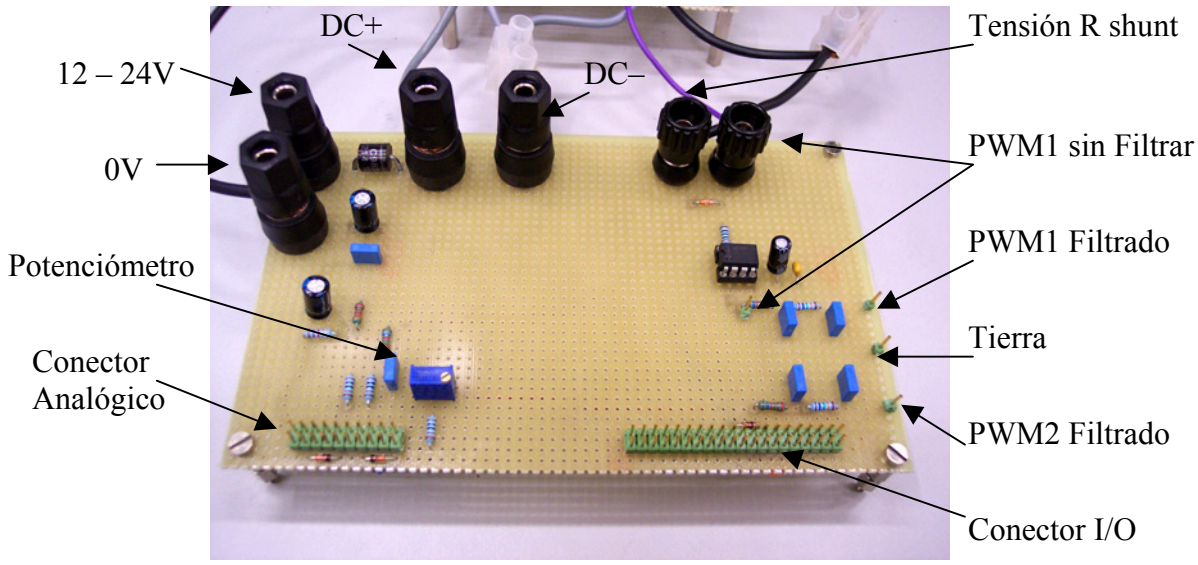


Figura 5.16. Placa de medidas y control.

Ambas placas se unen mediante cables de 1.5mm^2 de sección para soportar hasta 5A de intensidad. Esto cables se han conectado empleado bornas de conexión que facilitan las uniones además de permitir conectar fácilmente el motor, la fuente de alimentación y equipos de medida (osciloscopio, voltímetro y amperímetro). De esta manera se permite un montaje y desmontaje sencillo de los distintos equipos empleados en los experimentos. También se han instalado pines sueltos en las salidas filtradas y sin filtrar de las señales PWM generadas por el DSP (figura 5.16), de manera que sea fácil la conexión de una sonda de osciloscopio (figura 5.17).

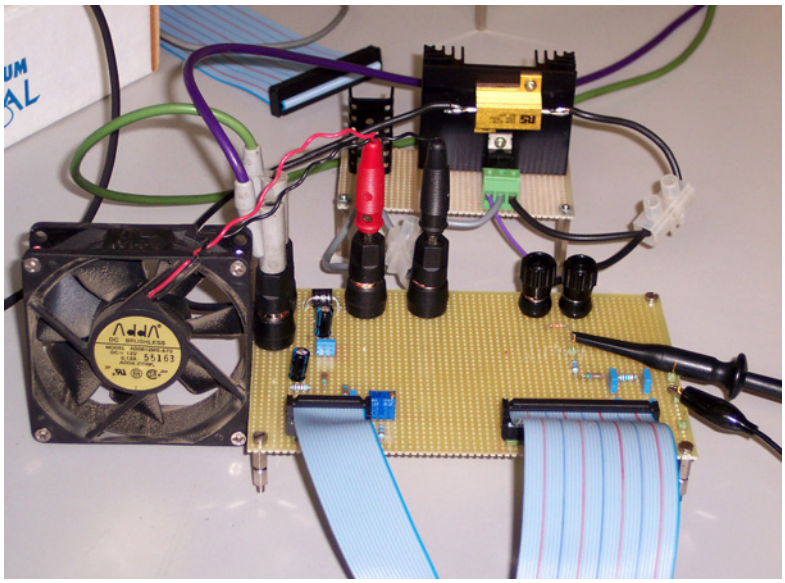
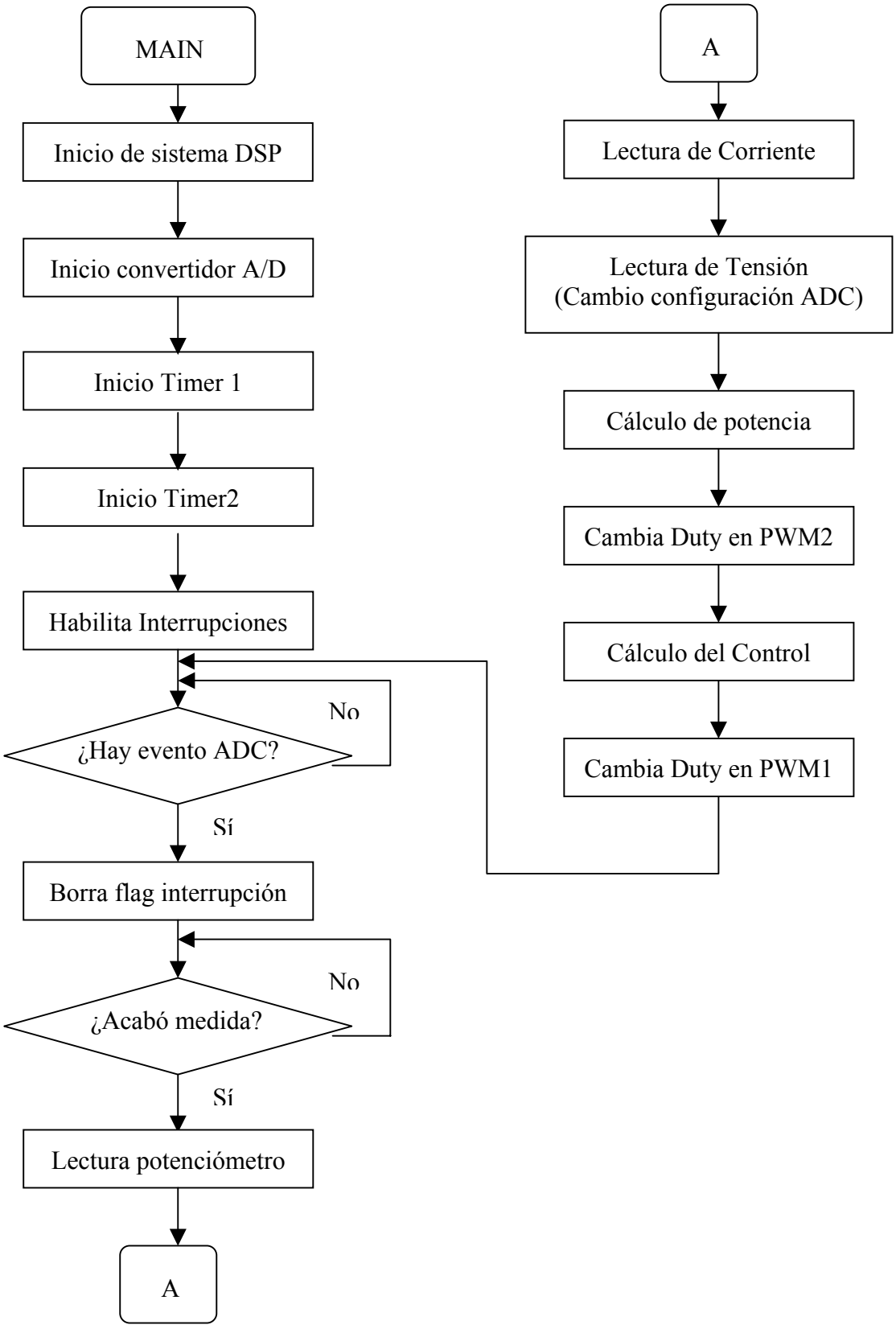


Figura 5.17. Montaje de pruebas con ventilador.

A la hora de poner a prueba la funcionalidad del código empleado y de la placa de pruebas construida, se optó por emplear un ventilador de 12V tomado de la fuente de alimentación de un PC. Esto se realizó así para evitar probar el circuito con intensidades altas mientras se comprobaba el correcto funcionamiento del sistema. Como se aprecia en el montaje de la figura 5.17, en ese momento no se habían instalado los componentes de protección detallados anteriormente (Zener y MAX4427), pero el sistema funcionaba correctamente tal como se esperaba. Cuando se decidió emplear el motor de 24V definitivo las primeras pruebas funcionaron correctamente, pero en una de ellas resultó dañada la señal PWM2 del DSP, que por aquel entonces era la señal empleada para el control del MOSFET. Después de analizar la situación, se concluyó que debía haberse producido una sobretensión en dicha señal que había provocado que la señal PWM dejara de ser perfectamente cuadrada (el flanco de bajada de la señal dejaba de ser vertical para asemejarse a la curva de descarga de un condensador), con lo que dejó de ser operativa para el control diseñado. Esta señal es compartida por el DSP con el pin 4 del puerto digital B de entrada/salida, que desde ese momento dejaba de funcionar como entrada, aunque sí funcionaba como salida. A raíz de estos acontecimientos se decidió emplear la señal PWM1 como señal de control, además de instalar en la placa los componentes de protección comentados con anterioridad. El Starter Kit que resultó dañado tiene el código “00/11-605” impreso en una pegatina situada en la cara de soldaduras de la placa.

5.3.3. Algoritmo empleado en la aplicación:

A continuación se detalla el algoritmo empleado en el código desarrollado para la aplicación del control de velocidad para un motor de continua.



5.3.4. Código empleado en la aplicación:

El código desarrollado para esta aplicación sigue lo indicado por el diagrama de flujo anterior. También se ha seguido la estructura empleada en los distintos proyectos de ejemplo detallados en capítulos anteriores. Por ello, los archivos empleados en esta aplicación son los siguientes:

- **Fichero System.h:**
  Archivo en el que se incluyen la definición de las etiquetas y la configuración de los bits de los registros de inicio del sistema DSP, como son: SCSR, IMR, IFR, WDCR, WSGR y la función para configurar éste último registro.

- **Fichero IO243.h:**
  Aquí se definen las etiquetas y configuración de bits de los registros de control de multiplexación OCRA y OCRB de las señales de entrada/salida del DSP.

- **Fichero ADC.h:**
  Detalla las etiquetas asignadas a los registros de control del convertidor A/D, además de definir la configuración de cada uno de sus bits.

- **Fichero EVA.h:**
  Incluye la definición de etiquetas de los registros de control de los temporizadores y de las interrupciones del Gestor de Eventos, así como la configuración de cada uno de sus bits.

- **Fichero System.c:**
  Archivo en el que se definen las funciones de iniciación del DSP, como son: configuración de estados de espera, función para atrapar interrupciones espúreas, configuración del Watchdog y configuración de interrupciones y de las máscaras de interrupción.

- **Fichero Dcmotor.c:**
  Archivo que contiene la función principal del programa, en la que se incluyen las funciones para el inicio de los temporizadores y convertidor ADC. También define las funciones encargadas de la lectura del potenciómetro, tensión de entrada e intensidad del motor, además de las funciones encargadas de generar las señales de potencia y control. Sigue la estructura del algoritmo detallado anteriormente.

- **Fichero vectors.asm:**
  Contiene la tabla de vectores de interrupción del DSP, que en el caso que nos ocupa únicamente emplea la interrupción asociada al inicio del DSP.

- **Fichero rts2xx.lib:**
  Librería de Code Composer que incluye la definición de distintas funciones de iniciación del entorno en lenguaje C y de otras funciones empleadas por el código diseñado.

- **Fichero Dcmotor.cmd:**
  Archivo con la distribución de memoria del DSP. Semejante a casos anteriores.

A continuación se comentan detalladamente los distintos archivos empleados en el proyecto, incluyéndose el listado completo de cada uno de ellos.

- **Fichero System.h:**

```
/*************************************************************/
/* system.h
   Archivo de definición de registros del sistema
/*
/*************************************************************/

#define SCSR *(volatile unsigned int *)0x7018      /* Registro de control del sistema */
#define IMR   *(volatile unsigned int *)0x0004     /* Registro de mascaras de
                                                                    interrupción */
#define IFR    *(volatile unsigned int *)0x0006    /* Registro de flags de interrupción */
#define WDCR *(volatile unsigned int *)0x7029 /* Registro de control del Watchdog */

/******************    SETUP del registro WDCR    ************************/
#define WDDIS                  1       /* 0 :Watchdog habilitado 1:deshabilitar  */
#define WDCHK2                 1       /* 0 : System reset   1: Normal OP        */
#define WDCHK1                 0       /* 0 : Normal Oper.      1: sysreset      */
#define WDCHK0                 1       /* 0 : System reset   1: Normal OP        */
#define WDSP                   7       /* Watchdog prescaler 7 : div 64          */
/*****************************************************************************/
/******************    SETUP del registro SCSR    ************************/
#define CLKSRC                 0       /* 0 : interno (20MHz)                    */
#define LPM                    0       /* 0 : Low power mode 0 if idle           */
#define ILLADR                 1       /* 1 : borra Illegal Address Flag         */
/*****************************************************************************/
/*********************    SETUP del registro IMR    ************************/
#define INT6                 0  /*  5 : Nivel INT6  enmascarado                 */
#define INT5                 0  /*  4 : Nivel INT5  enmascarado                 */
#define INT4                 0  /*  3 : Nivel INT4  enmascarado                 */
#define INT3                 0  /*  2 : Nivel INT3  enmascarado                 */
#define INT2                 0  /*  1 : Nivel INT2  enmascarado                 */
#define INT1                 0  /*  0 : Nivel INT1  enmascarado                 */
/*****************************************************************************/
/******************    SETUP del registro WSGR  ************************/
#define BVIS                   0       /* 10-9 : 00 Bus visibility OFF           */
#define ISWS                   0       /* 8 -6 : 000, 0 Estados de espera para IO */
#define DSWS                   0       /* 5 -3 : 000, 0 Estados de espera para data */
#define PSWS                   0       /* 2 -0 : 000, 0 Estados de espera para code */
/*****************************************************************************/

/* Definición de bits para configurar estados de espera */

#define IOWSB1       0x0080

/*  Definición de función MACRO para configurar registro WSGR */

#define _WSGR          0FFFFh     /* Definición de registro WSGR en espacio I/O */
```

```
#define STR(x) #x

#define OUTMAC(address,data)  \
 asm("       LDPK    _"STR(data));  \
 asm("       OUT     _"STR(data) "," STR(address))

void bad_trap(void);                    /* Función para atrapar interrupciones espúreas */
void init_system(void);                         /* Función de inicialización del sistema */

/* Fin de system.h */
```

Este archivo se encarga principalmente de la definición de las direcciones y configuración de los bits de los registros de inicialización del sistema: SCSR, IMR, IFR; WDCR y WSGR. También define una función macro que se encarga de configurar el registro WSGR que se mapea en la memoria I/O del sistema. Estas definiciones serán más tarde empleadas en el fichero "system.c".

- **Fichero IO243.h:**

```
/*****************************************************************/
/*  io243.h
  Archivo de definición de los registros de control de las señales de entrada/salida
*/
/*****************************************************************/

/* Registros de multiplexación entre función primaria y secundaria de los pines */

#define OCRA *(volatile unsigned int *) 0x7090
#define OCRB *(volatile unsigned int *) 0x7092

/*********************  SETUP del registro OCRA  **********************/
#define OCRA15                  0       /* 0 : IOPB7   1 : TCLKIN               */
#define OCRA14                  0       /* 0 : IOPB6   1 : TDIR                 */
#define OCRA13                  1       /* 0 : IOPB5   1 : T2PWM                */
#define OCRA12                  1       /* 0 : IOPB4   1 : T1PWM                */
#define OCRA11                  0       /* 0 : IOPB3   1 : PWM6                 */
#define OCRA10                  0       /* 0 : IOPB2   1 : PWM5                 */
#define OCRA9                   0       /* 0 : IOPB1   1 : PWM4                 */
#define OCRA8                   0       /* 0 : IOPB0   1 : PWM3                 */
#define OCRA7                   0       /* 0 : IOPA7   1 : PWM2                 */
#define OCRA6                   0       /* 0 : IOPA6   1 : PWM1                 */
#define OCRA5                   0       /* 0 : IOPA5   1 : CAP3                 */
#define OCRA4                   0       /* 0 : IOPA4   1 :CAP2/QEP2             */
#define OCRA3                   0       /* 0 : IOPA3   1 : CAP1/QEP1            */
#define OCRA2                   0       /* 0 : IOPA2   1 :XINT1                 */
#define OCRA1                   0       /* 0 : IOPA1   1 :SCIRXD                */
#define OCRA0                   0       /* 0 : IOPA0   1 : SCITXD               */
/*****************************************************************/
```

```
/*********************    SETUP del registro OCRB    *********************/
#define OCRB9                0       /* 0 : IOPD1   1 : XINT2/EXTSOC      */
#define OCRB8                1       /* 0 : CKLKOUT  1 : IOPD0            */
#define OCRB7                0       /* 0 : IOPC7   1 : CANRX             */
#define OCRB6                0       /* 0 : IOPC6   1 : CANTX             */
#define OCRB5                0       /* 0 : IOPC5   1 : SPISTE            */
#define OCRB4                0       /* 0 : IOPC4   1 : SPICLK            */
#define OCRB3                0       /* 0 : IOPC3   1 : SPISOMI           */
#define OCRB2                0       /* 0 : IOPC2   1 : SPISIMO           */
#define OCRB1                1       /* 0 : BIO     1 : IOPC1             */
#define OCRB0                1       /* 0 : XF      1 : IOPC0             */
/*****************************************************************/
```

En este archivo se indican las funciones que van a tener los distintos pines de entrada salida del DSP durante la aplicación. En particular se configuran los pines dedicados a las señales PWM1 y PWM2.

- **Fichero ADC.h:**

```
/*********************************************************/
/* adc.h Fichero de configuración del convertidor ADC    */
/*********************************************************/

#define ADCTRL1     *(volatile unsigned int *) 0x7032
#define ADCTRL2     *(volatile unsigned int *) 0x7034

#define ADCFIFO1    *(volatile unsigned int *) 0x7036
#define ADCFIFO2    *(volatile unsigned int *) 0x7038

/*******************  SETUP del registro ADCCTRL1  *******************/
#define SOFTFREE          2  /* 15-14 : 10 completa conversión antes de parar  */
#define ADCIMSTART        0  /* 13 :  no comienza conversión inmediatamente    */
#define ADC2EN            1  /* 12 :  unidad ADC2 habilitada                   */
#define ADC1EN            1  /* 11 :  unidad ADC1 habilitada                   */
#define ADCCONRUN         0  /* 10 :  conversión en modo no continuo           */
#define ADCINTEN          1  /* 9   :  habilita interrupción del ADC           */
#define ADCINTFLAG        1  /* 8   :  borra interrupciones previas            */
#define ADC2CHSEL         3  /* 6-4:  011 Canal 3 para unidad ADC2             */
#define ADC1CHSEL         0  /* 3-1:  000 Canal 0 para unidad ADC1             */
#define ADCSOC            1  /* 0   :  Inicio de conversión                    */
/*****************************************************************/
/*******************  SETUP del registro ADCCTRL2  *******************/
#define ADCIM             1  /* 14 : flag de interrupción del ADC              */
#define ADCINTPRI         0  /* 11 : prioridad de interrupción de ADC en nivel 6 */
#define ADCEVSOC          1  /* 10: habilitado inicio ADC por Gestor de Eventos */
#define ADCEXTSOC         0  /*  9 : deshabilita inicio externo del ADC        */
#define ADCPSCALE         7  /* 2-0: 000 prescaler CLKOUT/1                    */
/*****************************************************************/
```

Se encarga de definir los registros empleados por el convertidor ADC, además de la configuración que emplearán cada uno de sus bits.

- **Fichero EVA.h:**

```
/*********************************************************/
/* EVA.h  Fichero de configuracion del Gestor de Eventos  */
/*********************************************************/

#define GPTCON      *(volatile unsigned int *) 0x7400
#define T1CNT       *(volatile unsigned int *) 0x7401
#define T1CMPR      *(volatile unsigned int *) 0x7402
#define T1PR        *(volatile unsigned int *) 0x7403
#define T1CON       *(volatile unsigned int *) 0x7404

#define T2CNT       *(volatile unsigned int *) 0x7405
#define T2CMPR      *(volatile unsigned int *) 0x7406
#define T2PR        *(volatile unsigned int *) 0x7407
#define T2CON       *(volatile unsigned int *) 0x7408

#define EVIMRA      *(volatile unsigned int*) 0x742C
#define EVIMRB      *(volatile unsigned int *)0x742D
#define EVIMRC      *(volatile unsigned int *)0x742E
#define EVIFRA      *(volatile unsigned int *)0x742F
#define EVIFRB      *(volatile unsigned int *)0x7430
#define EVIFRC      *(volatile unsigned int *)0x7431

/******************      SETUP del registro GPTCON    *********************/
#define GPTCON_T2TOADC        0
                              /* 10-9  : T2TOADC = 00 : GPT2 no inicia ADC          */
#define GPTCON_T1TOADC        2
                            /*  8-7  : T1TOADC = 10 : GPT1  inicia ADC por periodo */
#define GPTCON_TCOMPOE        1
          /*   6   : TCOMPOE = 0  : habilitadas las 2 salidas de comparación del GPT */
#define GPTCON_T2PIN          1
                          /*  3-2  : T2PIN = 01 : Pol. de salida GPT2 = activa nivel bajo */
#define GPTCON_T1PIN          2
                           /*  1-0  : T1PIN = 10 : Pol. de salida GPT1 = activa nivel alto */
/*********************************************************************/
/*********************      SETUP del registro T1CON    *********************/
#define T1CON_FREESOFT        2
                    /* 15-14 FREE, SOFT : 10 libre en emulación JTAG suspendido    */
#define T1CON_TMODE           2
                         /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T1CON_TPS             1
                  /* 10-8  : TPS2-0     : 001  prescaler del reloj de entrada CPUCLK/2  */
#define T1CON_TENABLE         1      /*  6    : TENABLE    : 1 habilita GPT1      */
#define T1CON_TCLKS           0
                                    /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T1CON_TCLD            0
                               /* 3-2   : TCLD1,0    : 00 Recarga del timer 1 al llegar a 0 */
#define T1CON_TECMPR          1
                  /* 1     :  TECMPR     : 1 Habilita operación de comparación del timer */
/*********************************************************************/
```

```
/*********************     SETUP del registro T2CON    *********************/
#define T2CON_FREESOFT          2
                    /* 15-14 FREE, SOFT : 10 libre en emulación JTAG suspendido   */
#define T2CON_TMODE             2
                          /* 12-11 : TMODE1,0   : 10 Modo Continuous up en conteo */
#define T2CON_TPS               1
                   /* 10-8  : TPS2-0     : 001  prescaler del reloj de entrada CPUCLK/2  */
#define T2CON_TSWT1             0   /*  7: TSWT1: 0 usa bit TENABLE de GPT2 */
#define T2CON_TCLKS             0
                                  /* 5-4   : TCLKS1,0   : 00 Fuente de reloj: interno  */
#define T2CON_TCLD              0
                               /* 3-2   : TCLD1,0    : 00 Recarga del timer 2 al llegar a 0 */
#define T2CON_TECMPR            1
                /* 1     :  TECMPR     : 1 habilita operación de comparación del timer     */
#define T2CON_SELT1PR           0
                        /* 0       :  SELT1PR     :0      use su propio registro de periodo  */
/*****************************************************************************/
/*********************    SETUP del registro EVIMRA    *********************/
#define T1OFINT                 0  /* 10 : Timer 1 overflow interrupt              */
#define T1UFINT                 0  /*  9 : Timer 1 underflow interrupt             */
#define T1CINT                  0  /*  8 : Timer 1 compare interrupt               */
#define T1PINT                  0  /*  7 : Timer 1 period interrupt                */
#defineCMP3INT                  0  /*  3 : Compare 3 interrupt                     */
#define CMP2INT                 0  /*  2 : Compare 2 interrupt                     */
#define CMP1INT                 0  /*  1 : Compare 1 interrupt                     */
#define PDPINT                  0  /*  0 : Power Drive Protect Interrupt           */
/*****************************************************************************/
/*********************    SETUP del registro EVIMRB    *********************/
#define T2OFINT                 0  /*  3 : Timer 2 overflow interrupt              */
#define T2UFINT                 0  /*  2 : Timer 2 underflow interrupt             */
#define T2CINT                  0  /*  1 : Timer 2 compare interrupt               */
#define T2PINT                  0  /*  0 : Timer 2 period interrupt                */
/*****************************************************************************/
/*********************    SETUP del registro EVIMRC    *********************/
#define CAP3INT                 0  /*  2 : Capture Unit 3 interrupt                */
#define CAP2INT                 0  /*  1 : Capture Unit 2 Interrupt                */
#define CAP1INT                 0  /*  0 : Capture unit 1 interrupt                */
/*****************************************************************************/
```

Archivo en el que se definen los distintos registros encargados del control de los temporizadores y de las interrupciones del Gestor de Eventos. En particular, se configuran ambos temporizadores para modo de cuenta continuo ascendente, siendo la polaridad del temporizador 1 activa a nivel alto y la del temporizador 2 activa a nivel bajo. También se enmascaran todas las posibles interrupciones del Gestor de Eventos, ya que no vana ser empleadas.

Los temporizadores se han configurado para generar ondas PWM de 10kHz de frecuencia. Este valor se ha elegido debido a que la frecuencia de conmutación debe ser mucho mayor que la de giro del motor (en nuestro caso 1300 rpm). Un valor típico es de 10 a 20kHz. Para accionar motores suelen emplearse altas frecuencias para evitar que se genere ruido en el motor en la banda de frecuencias audible por las personas.

- **Fichero System.c:**

```
/*********************************************************************/
/* Funciones para el inicio de sistema DSP                           */
/*********************************************************************/

#include "system.h"
#include "eva.h"


/* Definición de funciones estáticas empleadas en este módulo */

static void set_system(void);
static void set_wait(void);

/* Variable global */

volatile unsigned int configdata;


/* Dos funciones empleadas por el programa principal */

void bad_trap(void)
{
 while(1);              /* Función para atrapar interrupciones espúreas */
}

void init_system(void)              /* Función de inicio del sistema */
{
 set_wait();                        /* Configura los estados de espera para I/O */

 set_system();                      /* inicia registros del sistema y de interrupción */

}


/*********************************************************************/
/*                          Subrutinas                               */
/*********************************************************************/

static void set_wait(void)
{
 configdata = IOWSB1;               /* Configura 2 estados de espera para I/O */
 OUTMAC( _WSGR, configdata);
}
```

```
/* Función de configuración del inicio del sistema */

static void set_system(void)
{
 asm (" setc INTM");          /* Deshabilita todas las interrupciones */
 asm (" clrc SXM");           /* Borra el bit de extensión de signo */
 asm (" clrc OVM");           /* Borra bit de modo de desbordamiento */
 asm (" clrc CNF");           /* Configura B0 como memoria de datos */

 WDCR=((WDDIS<<6)+(WDCHK2<<5)+(WDCHK1<<4)+(WDCHK0<<3)+WDSP);
                                          /* Inicializa el registro WDCR */

 SCSR = ((CLKSRC<<14)+(LPM<<12)+ILLADR);        /* Inicializa SCSR     */

 EVIMRA=((T1OFINT<<10)+(T1UFINT<<9)+(T1CINT<<8)+(T1PINT<<7)+
                    (CMP3INT<<3)+(CMP2INT<<2)+(CMP1INT<<1)+(PDPINT));
                                  /* Registro de máscaras del grupo A del EV  */

 EVIMRB=((T2OFINT<<3)+(T2UFINT<<2)+(T2CINT<<1)+(T2PINT));
                                  /* Registro de máscaras del grupo B del EV  */

 EVIMRC=((CAP3INT<<2)+(CAP2INT<<1)+(CAP1INT));
                                  /* Registro de máscaras del grupo C del EV  */

 EVIFRA=0xFFFF;     /* Registro de flags del grupo A del EV  */

 EVIFRB=0xFFFF;     /* Registro de flags del grupo B del EV  */

 EVIFRC=0xFFFF;     /* Registro de flags del grupo C del EV  */

IMR=((INT6<<5)+(INT5<<4)+(INT4<<3)+(INT3<<2)+(INT2<<1)+(INT1));
                                      /* Registro de máscaras de interrupción */

 IFR=0xFFFF;          /* Borra todas las interrupciones pendientes */

}
```

En este archivo se agrupan la configuración de los registros de inicio del sistema, tomando la misma estructura que la empleada en ejemplos anteriores, aunque en este caso se ha decidido que fuesen incluidos en un archivo aparte para mayor claridad del código debido a su extensión. En él se definen dos funciones: una para la configuración de los estados de espera del espacio I/O, y otra para la inicialización del DSP. En esta última se incluyen los registros de inicio habituales del sistema (SCSR, WDCR) además de la configuración de máscaras de interrupción del Gestor de Eventos y del sistema, así como del borrado de los distintos flags indicadores de interrupciones pendientes.

- **Fichero Dcmotor.c:**

```
/*********************************************************************/
/*                                dcmotor.c                          */
/*********************************************************************/

/*
  Control de un motor DC controlado por T1PWM.
  Monitorización de tensión de alimentación en ADCIN5.
  Monitorización de corriente del motor en ADCIN3.
  Calcula potencia tomada por el motor y produce salida en T2PWM.
*/

#include "system.h"
#include "eva.h"
#include "io243.h"
#include "adc.h"

/* Inicio del Temporizador 1. */

void init_GPT1(void)
{
OCRB = ((OCRB9<<9)+(OCRB8<<8)+
                (OCRB7<<7)+(OCRB6<<6)+(OCRB5<<5)+(OCRB4<<4)+
                (OCRB3<<3)+(OCRB2<<2)+(OCRB1<<1)+OCRB0);
                                                    /* Inicializa OCRB */


 OCRA = ((OCRA15<<15)+(OCRA14<<14)+(OCRA13<<13)+(OCRA12<<12)+
       (OCRA11<<11)+(OCRA10<<10)+(OCRA9<<9)+(OCRA8<<8)+
       (OCRA7<<7)+(OCRA6<<6)+(OCRA5<<5)+(OCRA4<<4)+
       (OCRA3<<3)+(OCRA2<<2)+(OCRA1<<1)+OCRA0);
                                                    /* Inicializa OCRA */
GPTCON=((GPTCON_T2TOADC<<9)+(GPTCON_T1TOADC<<7)+
            (GPTCON_TCOMPOE<<6)+(GPTCON_T2PIN<<2)+(GPTCON_T1PIN));
                                  /* Inicia el registro de control del temporizador */

T1CON=((T1CON_FREESOFT<<14)+(T1CON_TMODE<<11)+(T1CON_TPS<<8)+
                (T1CON_TCLKS<<4)+(T1CON_TCLD<<2)+(T1CON_TECMPR<<1));
                               /* Configura el registro de control del temporizador 1 */

 T1PR    = 1000;              /* Señal de 10 kHz en F243*/
 T1CMPR  = 1000;              /* Duty cycle para motor parado en inicio */
 T1CNT   = 0xFFFE;            /* Valor inicial de contador a -2 */

}
```

```
/* Inicio del Temporizador 2 */

void init_GPT2(void)
{

T2CON=((T2CON_FREESOFT<<14)+(T2CON_TMODE<<11)+(T2CON_TPS<<8)+
                (T2CON_TSWT1<<7)+(T2CON_TCLKS<<4)+(T2CON_TCLD<<2)+
                                    (T2CON_TECMPR<<1)+T2CON_SELT1PR);
                                /* Configura el registro de control del temporizador 2 */

 GPTCON =GPTCON + (GPTCON_T2PIN<<2);          /* Activo a nivel bajo*/

 T2PR    = 1000;              /* Señal de 10 kHz en F243*/
 T2CMPR  = 0;                 /* Duty = 0% */
 T2CNT   = 0xFFFE;            /* Valor inicial de contador a -2 */

}


 /*********************************************************/
/*
  Inicio del convertidor analógico-digital.
  Se configura para realizar medidas en canal 0 y 3.
  Comienza conversión por evento de periodo en Temporizador 1.
*/
/*********************************************************/


void init_ADC()
{

ADCTRL1= ((SOFTFREE<<14)+(ADCIMSTART<<13)+(ADC2EN<<12)+
                        (ADC1EN<<11) +(ADCCONRUN<<10) +(ADCINTEN<<9)+
                        (ADCINTFLAG<<8)+(ADC2CHSEL<<4)+(ADC1CHSEL<<1));
                                /*  Configura registro de control 1 del ADC */

 ADCTRL2= ((ADCIM<<14)+(ADCINTPRI<<11)+
        (ADCEVSOC<<10)+(ADCEXTSOC<<9)+ADCPSCALE);
                                /*  Configura registro de control 2 del ADC */
}


/*********************************************************/
/*
  Lectura del potenciómetro en ADCIN0.
  Valor medido en ADCFIFO1 varía en el rango de 0 a FFC0h.

  Para asegurarse que el valor es positivo se desplaza un bit a la derecha
  el resultado de la medida. Esto hace que el valor se encuentre
  en el rango de 0 a 7FE0h.
```

```
 Se multiplica la medida por un factor de escala para generar
 un valor que oscile entre 0 y 100%

 Se devuelve un valor entre 0 y 100%.
*/
/*********************************************************/

signed int read_potentiometer(void)
{
 signed int return_value;              /* Declaración de variables empleadas */
 signed long scaled;

 scaled = (ADCFIFO1 >> 1);             /* Hace sitio al bit de signo */

 scaled *= 0x00C8;     /* Multiplica por un factor de escala para rango entre  0 y 100 */

 return_value = (signed int)(scaled >> 16);

 return(return_value);                  /* Devuelve el valor escalado */
}

/*********************************************************/
/* Control del motor.

 Entrada: 0 a 100% del potenciómetro .
 Salida: 0 a 1000 para el ancho del pulso PWM.

 Para asegurarse de que el motor está completamente parado o
 a su máxima velocidad, se ignora un 5% del rango de variación.          */
/*********************************************************/

signed int control_motor(signed int input)
{
 signed int return_value;              /* Declaración de variables empleadas */
 signed long scaled;

 if ( input < 5)          /* Si potenciómetro está a menos del 5% de su rango */
  {
  return_value = 0;             /* se devuelve ancho de PWM que detiene el motor */
  }
 else if ( input > 95)             /* Si potenciómetro está a más del 95% de su rango */
  {
  return_value = 1000;                  /* se devuelve ancho de PWM que acelera
                                                        totalmente el motor */
  }
 else                      /* Si potenciómetro está entre 5% y 95% */
  {                                /* calcula el ancho de PWM entre 0 y 1000*/

  scaled = (long)(input - 5);           /* Ignora un 5% del rango de variación */

  scaled *= 2845;             /* Multiplica por factor de escala para rango de 0 a 1000 */
```

```
  return_value = (signed int)(scaled >> 8);
  }

 /* Como la variación del potenciómetro es inversa al valor de comparación
    del pulso PWM, un valor alto en el mismo se corresponde con
    un valor bajo del registro comparador T1CMPR
*/

 return(1000 - return_value);                    /* Devuelve valor del ancho PWM */
}

/*************************************************************/
/*
  Medida de la corriente del motor.
  Lectura del canal de entrada ADCIN3.
  El valor almacenado en ADCFIFO2 es proporcional a la corriente.
  Corriente máxima = 5V / 0.33R = 15.15A
  Por tanto, el valor máximo en ADCFIFO2 (0xFFC0 ) representará a  15.15A.
  Esta función devuelve un valor entre 0 y 1515 (mejora errores de redondeo).
*/
/*************************************************************/

volatile signed int measure_current(void)
{
 signed int return_value;                /* Define variables empleadas */
 signed long scaled;

 scaled = (ADCFIFO2 >> 1);               /* Hace sitio al bit de signo */

 scaled *= 0x0BD8;              /* Multiplica por factor de escala para dar valor entre
                                                          0 y 1515 en decimal */

 return_value = (signed int)(scaled >> 16);

 return(return_value);                           /* Devuelve valor escalado */
}

/*****************************************************************/
/*
   Medida de la tensión de alimentación a través de divisor de tensión.
   Lectura de canal de entrada ADCIN5.
   Es necesario cambiar configuración del convertidor ADC para lectura del canal.
   El valor en ADCFIFO2 es proporcional a la tensión.

   La tensión máxima será = 5V * (33k + 3.3k)/3.3k = 55V
   Por tanto, el valor máximo en ADCFIFO2 (0xFFC0) representa 55V
   La función devuelve un valor entre 0 y 5500
   para representar la tensión de 0V a 55V (mejora errores de redondeo).
*/
/*************************************************************/
```

```
volatile signed int measure_voltage(void)
{

 signed int return_value;              /* Definición de variables empleadas */
 signed long scaled;
 signed int x;
 x=0;


/* Cambio de la configuración del registro ADCTRL1 para leer el canal 5 con ADC2 */

 ADCTRL1=((ADCTRL1 & 0xFF8F)|(0x0005<<4));

 ADCTRL1|=0x0100;                      /* Borra flag de interrupción pendiente */

 asm (" LACL 7038h");                  /* Vaciado del la pila ADCFIFO2 para eliminar
 asm (" LACL 7038h");                        valores almacenados de otros canales */
 asm (" LACL 7038h");


 while((ADCTRL1 & 0x0100)==0)          /* Bucle de espera mientras se genera
 x++;                                        una nueva interrupción */

 x=0;

 while(ADCTRL1 & 0x0080)               /* Bucle de espera mientras finaliza
 x++;                                        la medida en el canal 5 */

 scaled = (signed long)(ADCFIFO2 >> 1);          /* Hace sitio al bit de signo */

 scaled *= 0x2B02;    /* Multiplica por factor de escala para dar valor entre  0 y 5500 */


/* Vuelta al estado inicial del registro ADCTRL1 para leer el canal 3 con ADC2 */

 ADCTRL1=((ADCTRL1 & 0xFF8F)|(0x0003<<4));

 ADCTRL1|=0x0100;                      /* Borra flag de interrupción pendiente */

 asm (" LACL 7038h");                  /* Lecturas de ADCFIFO2 para eliminar
 asm (" LACL 7038h");                        valores almacenados de otros canales */
 asm (" LACL 7038h");


 return_value = (signed int)(scaled >> 16);

 return(return_value);                 /* Devuelve valor escalado */

}
```

```
/*********************************************************/
/*Calcula potencia absorbida por el sistema.
  Potencia máxima = 15.15 A * 55 V
  Se limita la escala de potencia a 50.00 V * 1.00A = 50 W.
  para su representación a escala en un osciloscopio entre 0 y 5V.
  La función devuelve un valor entre 0 y 1000 para el ancho de T2PWM
  y representar la potencia de 0 a 50.0W.                          */
/*********************************************************/

volatile signed int calculate_power(signed int current, signed int voltage)
{
 signed long power;                  /* Definición de variables empleadas */
 signed int return_value;

 power = current * voltage;          /* cálculo de la potencia */

 power *= 131;       /* Multiplica por factor de escala adecuado para rango de 0 a 50 */

 return_value = (signed int)(power >> 16);

 if ( return_value > 1000)           /* Limita salida de potencia a 50 W */
  {
   return_value = 1000;                      /* Devuelve valor escalado */
  }

 return (return_value);                      /* Devuelve valor escalado */
}


/*********************************************************/
/*
  Programa principal de control del motor.

  Emplea los temporizadores a 10 kHz para producir señales PWM
  con la entrada leída del potenciómetro.
*/
/*********************************************************/

void main(void)
{
 signed int x,y;              /* Variables de usos general */
 signed int voltage;          /* Tensión de alimentación del motor */
 signed int current;          /* Corriente tomada por el motor */

 init_system();               /* Inicialización de variables y del DSP */
 init_ADC();                  /* Inicialización del ADC */
 init_GPT1();                 /* Inicialización del Temporizador 1 */
 init_GPT2();                 /* Inicialización del Temporizador 2 */

 asm (" clrc INTM");          /* Habilitación global de las interrupciones */
```

```
 for ( ; ; )                    /* Bulce sin fin */
  {

  if ( ADCTRL1 & 0x0100)        /* Comprueba si ha habido evento en el ADC */
   {
    ADCTRL1 |= 0x0100;          /* Borra el flag del evento ADC */

    x = 0;

    while (ADCTRL1 & 0x0080)
    x++;                         /* Espera que acabe la medida del ADC*/

    x = read_potentiometer();   /* Lectura del potenciómetro en ADCIN0 */

    current = measure_current(); /* Lectura de la corriente: 0 a 15.15 A */

    voltage = measure_voltage(); /* Lectura de la tension: 0 a 55 V */

    y = calculate_power(current, voltage);  /* Cálculo de la potencia */

    T2CMPR = y;      /* Cambio del ancho del pulso de señal de potencia T2PWM */

    y = control_motor(x);       /* Cálculo del control de velocidad del motor */

    T1CMPR = y;          /* Cambio del ancho del pulso de control T1PWM */
   }

  }

}
```

En este archivo se inician los distintos periféricos empleados en la aplicación, como son los temporizadores y el convertidor ADC. Los temporizadores son configurados para generar ondas PWM de 10kHz de frecuencia, como se comentó con anterioridad. El temporizador 1 es configurado como activo a nivel alto, por lo que se toma 1000 como valor inicial del comparador, de manera que inicialmente el motor esté detenido completamente. Dado que la comparación se produce al final del periodo, para aumentar la velocidad y por tanto el ancho del pulso, debe disminuirse el valor del comparador a medida que aumenta el valor de tensión dada por el potenciómetro (figura 5.18). De esta manera se modifica la posición del flanco de subida de la señal T1PWM que controla la velocidad del motor, mientras que el flanco de bajada al final del periodo permanece invariante. De igual forma, el temporizador 2 es configurado para la misma frecuencia, pero con polaridad activa a nivel bajo. Por ello el valor inicial del comparador cuando el motor está detenido es 0. Al darse la comparación al inicio del periodo, para aumentar la señal de potencia T2PWM debe aumentarse el valor del registro comparador, con lo que se modifica la posición del flanco de bajada de la señal, permaneciendo constante el flanco de subida (figura 5.18).

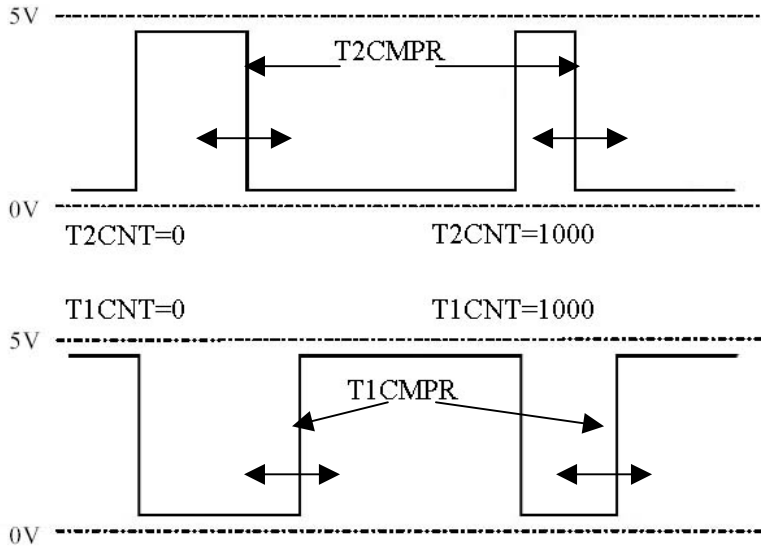


Figura 5.18. Control del acho de los pulsos PWM.

Por otro lado, se inicializa el convertidor analógico-digital para que se realice las medidas cuando el temporizador 1 llegue a su periodo, generándose una señal de interrupción. En un principio se configura el convertidor para leer los canales ADCIN0 y ADCIN 3 (potenciómetro e intensidad), pero es necesario cambiar esta configuración a la hora de medir la tensión de alimentación. Esto se realiza en la función definida para ese objetivo "measure_voltage", en la que se cambia la configuración del registro de control ADCTRL1. Para que tenga efecto el cambio es necesario vaciar la pila ADCFIFO2 de valores leídos con la configuración anterior. Ello se realiza simplemente leyendo en varias ocasiones dicho registro, ya que este consta de dos niveles de almacenamiento, con lo que se van sacando los sucesivos valores almacenados. También es necesario borrar el indicador de interrupción del ADC y esperar a que se realice una nueva medida con la nueva configuración, hecho lo cual se procede a leer el nuevo valor. Para finalizar, debe establecerse el estado inicial del registro ADCTRL1 para posibilitar la correcta medición de las otras variables del sistema. Este método empleado se debe a la poca versatilidad que presenta el convertidor del TMS320F243, que solo permite realizar dos medidas en dos canales de forma simultánea. Por ello existe un pequeño desfase de tiempo entre la realización de las distintas medidas de variables.

A la hora de emplear las medidas realizadas, es necesario el escaldo de las mismas para la obtención de valores representativos de las magnitudes a las que representan. Para ello es necesario multiplicar las medidas por un factor de escala adecuado al rango de valores que se quiere tomar. Esto se realiza como se detalló en el capítulo 4 cuando se analizó en funcionamiento del convertidor ADC. Los factores de escala empleados en la aplicación son los siguientes:

$$Factor\_escala|_{Potenciómetro} = \frac{0x64 \cdot 2^{16}}{0x7FE0} = 0x00C8$$

$$Factor\_escala|_{Control} = \frac{1000 \cdot 2^{8}}{90} = 2845$$

$$Factor\_escala|_{Corriente} = \frac{0x05EB \cdot 2^{16}}{0x7FE0} = 0x0BD8$$

$$Factor\_escala|_{Tensión} = \frac{0x157C \cdot 2^{16}}{0x7FE0} = 0x2B02$$

$$Factor\_escala|_{Potencia} = \frac{1000 \cdot 2^{16}}{500000} = 131$$

Para el factor de escala del potenciómetro se toma como valor máximo del rango deseado 100 (0x064); para el cálculo de potencia se ha aplicado una regla de tres para una salida máxima de 1000 cuando el rango de variación del porcentaje es de 90 (descontando 5% por los márgenes superior e inferior y únicamente se desplazan 8 bits para obtener un número menor de 65536); en la medida de la intensidad se toma como valor máximo 1515 (0x05EB); en la tensión se toma valor máximo 5500 (0x157C) y, por último, en la potencia el valor máximo asignado es 5000V*100A = 500000 (50W) para una salida de comparación de 1000. Salvo en uno de los casos, en todas las funciones de medidas analizadas los valores almacenados son desplazados 16 bits a la dercha antes de ser devueltos a la función principal. Ello se debía a que el registro multiplicador es de 32 bits y que los valores leídos de las FIFIOs se encuentran almacenados en los 10 bits más significativos de las mismas. De esta manera se reducían los errores de redondeo en las operaciones, por lo que es necesario desplazar los resultados para obtener los valores escalados.

Como se detalló en el algoritmo de apartados anteriores, el programa principal espera a que se produzca un arranque del convertidor ADC por el Gestor de Eventos, para luego comenzar a realizar las mediciones cuando éstas han finalizado. Con estos valores se procede al cálculo del Duty Cycle adecuado para ese instante de las ondas PWM de control y de medida de potencia.

- **Fichero vectors.asm:**

Archivo que contiene las funciones asociadas a las distintas interrupciones que pueden producirse en el DSP. En este caso no se emplea ninguna salvo la de inicio del sistema “c_int0”.

```
        .ref   _bad_trap   ; función para atrapar interrupciones ilegales
        .ref    _c_int0    ; punto de entrada al código

        .global _vector

        .sect   "vectors"
_vector:
RSVECT      B   _c_int0    ; PM 0  Reset Vector          1
INT1        B   _bad_trap ; PM 2 Int level 1             4
INT2        B   _bad_trap ; PM 4 Int level 2             5
INT3        B   _bad_trap ; PM 6 Int level 3             6
INT4        B   _bad_trap ; PM 8 Int level 4             7
INT5        B   _bad_trap ; PM AInt level 5              8
INT6        B   _bad_trap ; PM CInt level 6              9
RESERVED    B   _bad_trap  ; PM E  (Analysis Int)       10
SW_INT8     B   _bad_trap ; PM 10         User S/W int   -
SW_INT9     B   _bad_trap ; PM 12         User S/W int   -
```

```
SW_INT10    B    _bad_trap ; PM 14        User S/W int   -
SW_INT11    B    _bad_trap ; PM 16        User S/W int   -
SW_INT12    B    _bad_trap ; PM 18        User S/W int   -
SW_INT13    B    _bad_trap ; PM 1A        User S/W int   -
SW_INT14    B    _bad_trap ; PM 1C        User S/W int   -
SW_INT15    B    _bad_trap ; PM 1E        User S/W int   -
SW_INT16    B    _bad_trap ; PM 20        User S/W int   -
TRAP        B    _bad_trap ; PM 22        Trap vector    -
NMI         B    _bad_trap ; PM 24        Non maskable int
EMU_TRAP    B    _bad_trap ; PM 26        Emulator Trap2
SW_INT20    B    _bad_trap ; PM 28        User S/W int   -
SW_INT21    B    _bad_trap ; PM 2A        User S/W int   -
SW_INT22    B    _bad_trap ; PM 2C        User S/W int   -
SW_INT23    B    _bad_trap ; PM 2E        User S/W int   -

      .end
```

- **Fichero Dcmotor.cmd:**

```
                    /* linker commands */
-c                  /* ROM autoinitialization */
-x                   /* force rereading libraries */
-o  motor.out       /* output file */
-m motor.map        /* map file */


                          /* specify memory map */
MEMORY
{
 PAGE 0 :                              /* program memory */
       VECS: origin = 00000h, length = 0003Fh
       CODE: origin = 00040h, length = 00DC0h
 PAGE 1 :                              /* data memory */
       Ext_Ram : origin = 08000h, length = 01000h
}

                          /* specify sections */
SECTIONS
{
 vectors   :   > VECS PAGE = 0
 .text     :   > CODE PAGE = 0
 .switch   :   > CODE PAGE = 0
 .data     :   > Ext_Ram PAGE = 1
 .bss      :   > Ext_Ram PAGE = 1
 .heap     :   > Ext_Ram PAGE = 1
 .stack    :   > Ext_Ram PAGE = 1
}
```

5.3.5. Procedimientos realizados:

Para poner en práctica la aplicación desarrollada es necesario tomar una serie de precauciones para evitar dañar algunos de los dispositivos, especialmente el DSP. Algunas de estas precauciones son simplemente comprobar que los niveles de tensión e intensidad de la fuente de alimentación empleada son los correctos, así como realizar las conexiones del montaje sin aplicarle tensión. También es necesario asegurarse de que los jumpers de configuración del DSP permanecen en las posiciones indicadas en capítulos anteriores, de modo que pueda emplearse con el software Code Composer y que las tensiones de referencia del convertidor ADC se toman de la misma placa. Esto último se comenta porque en un futuro puede ser conveniente emplear señales de referencia externas para el convertidor.

Una vez montado y conexionado el sistema es necesario seguir un determinado orden en el encendido de los distintos componentes del mismo. Esto se realiza para asegurarse de que ningún componente resulta dañado. En particular este procedimiento intenta impedir que de forma accidental se aplique tensión a señales del DSP que trabajan como salidas, lo que podría destruir el dispositivo. Por ello se recomienda proceder de la siguiente manera:

1) Conexionado de los aparatos de medida (voltímetro, osciloscopio, etc.).
2) Conexión de la alimentación del DSP.
3) Ejecución de Code Composer.
4) Carga del proyecto realizado y compilación del mismo.
5) Descarga del archivo compilado al DSP.
6) Ejecución del código en el DSP para la configuración del mismo y de las señales empleadas en la aplicación como entradas o salidas.
7) Encendido de las fuentes de alimentación empleadas. Si se emplean varias, encender la de alimentación principal en último lugar.

Una vez realizados estos pasos el sistema está plenamente operativo, con lo que se pueden realizar las distintas pruebas que se crean convenientes. Destacar que, de igual forma que anteriormente, el proceso de parada del sistema debe seguir los mismos pasos pero en orden inverso, es decir:

1) Apagado de las distintas fuentes de alimentación, empezando por la de alimentación principal al sistema.
2) Detención del código cargado en el DSP.
3) Cierre del software Code Composer.
4) Desconexión de la alimentación del DSP.

Siguiendo esto pasos se evitan en gran medida accidentes que puedan provocar daños en el DSP principalmente, ya que este dispositivo es muy sensibles a sobretensiones en sus señales de entrada/salida. Se ha tomado este proceso de inicio y parada del proceso debido a que, mientras no se realice un reset del dispositivo, es posible que se mantengan las configuraciones de aplicaciones descargadas anteriormente al DSP. Esto puede provocar que señales que van a emplearse como salidas y estén conexionadas como tal, puedan estar configuradas como entradas por aplicaciones empleadas con anterioridad, con el consiguiente peligro de destrucción de las mismas.

5.3.5.1. Pruebas iniciales con ventilador:

En un principio, y para evitar en lo posible producir daños al circuito y al DSP, se optó por emplear un motor de continua de menor tensión e intensidad que el elegido como objetivo del proyecto. Se tomó como motor DC un ventilador procedente de la fuente de alimentación de un PC que trabaja a 12V y 0.12A (figura 5.17), evitando así trabajar con valores elevados de tensión e intensidad para realizar las primeras pruebas de la placa diseñada. Con este montaje era necesario emplear una fuente de alimentación de 12V como se muestra en la figura 5.19.

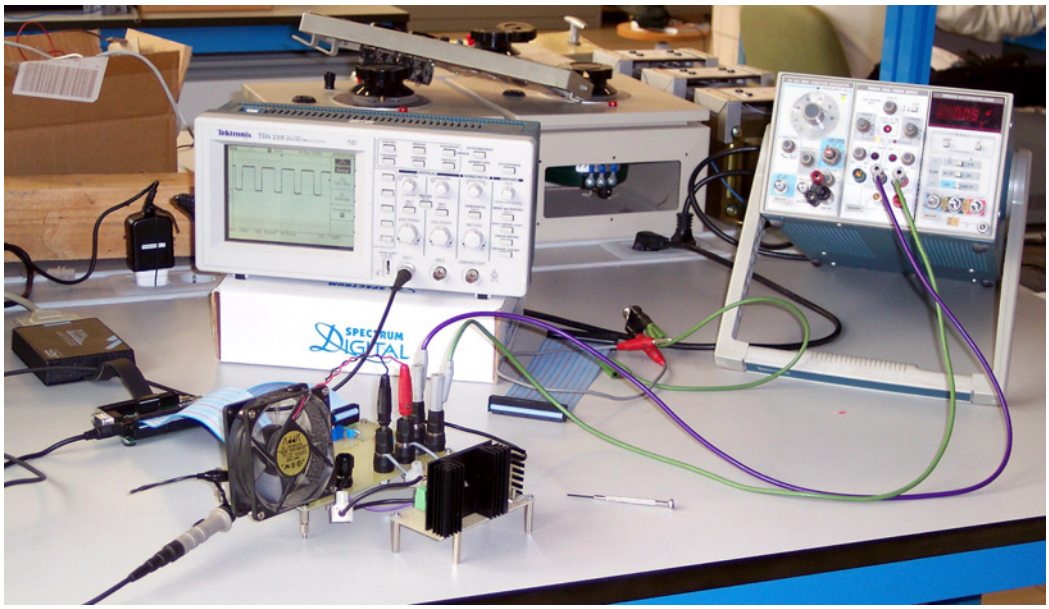


Figura 5.19. Montaje con ventilador.

Las primeras pruebas realizadas fueron satisfactorias en un primer momento, ya que se disparaba correctamente el transistor y la variación de los pulsos PWM se producían de forma inmediata después de manipular el potenciómetro. El control de la velocidad del ventilador mediante el mismo se realiza girando el potenciómetro en el sentido anti-horario para aumentar la velocidad, y en sentido horario para disminuirla.

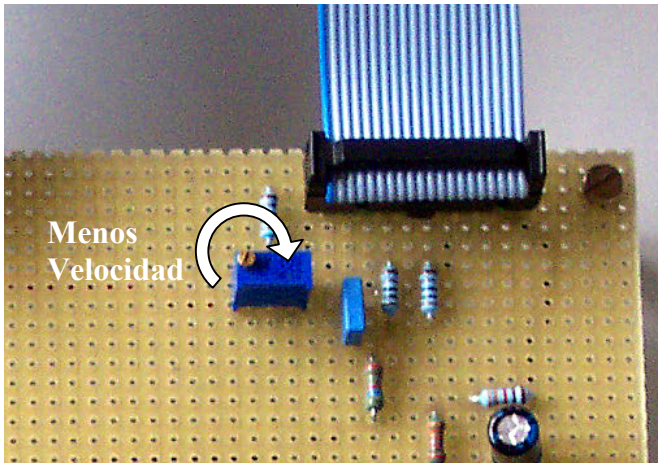


Figura 5.20. Variación de velocidad con potenciómetro.

Aunque en un principio todo indicaba que el montaje era correcto, se observó un problema en la señal que reproducía la potencia absorbida por el sistema. En concreto la potencia siempre era cero. Se comprobó el funcionamiento del código empleado con la placa de pruebas desarrollada para los ejemplos del capítulo 4, para lo que hubo que incluir un tercer potenciómetro en el canal 5 y, por comodidad, se cambió el potenciómetro que inicialmente estaba en el canal 1 al canal 3. De esta forma, con los tres potenciómetros instalados se simulaban las medidas en los tres canales empleados en la placa de control, y que representaban al potenciómetro de control de velocidad, a la medida de intensidad del motor y a la de tensión de alimentación del sistema. Esta prueba concluyó que el código era correcto, por lo que debía deberse a un problema en el circuito. Para comprobar esto último se decidió emplear el programa de desarrollo VisSim, el cual permite leer fácilmente los canales analógicos y representarlos gráficamente. Ello dio como resultado que el canal encargado de medir la intensidad del motor mediante la tensión de la resistencia Shunt medía continuamente cero. Se procedió a medir la tensión de entrada a dicho canal con un voltímetro, y se comprobó que la tensión a la entrada del mismo era negativa si se tomaba como referencia de tierra los pines GND del convertidor. Como el convertidor solo puede medir tensiones entre 0 y 5V, el resultado de la medida siempre se daba como cero, produciendo una señal de potencia de igual valor. Por el contrario, si se tomaba como referencia de tierra la de la fuente de alimentación para medir la tensión de entrada al canal, resultaba que la tensión era positiva y muy próxima a lo que debía de medir el canal. Era claro que el problema residía en que las distintas medidas realizadas empleaban niveles de referencia de tierras distintas. Por ello, se decidió conectar ambas tierras para unificarlas, dando como resultado la corrección del problema en la señal de potencia..

Una vez corregido el problema de la señal de potencia, al estar preparado el código para un motor de potencia mayor, era necesario modificar el factor de escala empleado en el código para generar una señal que oscilase en una escala de valores adecuada. Por ello se modificó el código descrito anteriormente tomando como factor de escala para la potencia:

$$Factor\_escala|_{Potencia} = \frac{1000 \cdot 2^{16}}{18000} = 3641$$

donde la potencia consumida por el sistema es P = 0.15A*12V = 1.8W. Se alimentó al ventilador con una intensidad de 0.15A para forzarlo ligeramente y que la velocidad máxima que alcanzase fuese sensiblemente superior al resto de velocidades en otros instantes.

Una vez efectuadas las correcciones detalladas se procede a la puesta en marcha del sistema siguiendo los pasos analizados en el apartado anterior. Mediante voltímetros se comprobó que las medidas de tensión en el divisor de tensión y de intensidad en la resistencia shunt daban como resultado un valor de potencia muy próximo al producido por la señal de potencia:

$$V_{canal3} \approx 0.05V \rightarrow I_{ventilador} = 0.33 * 0.05 \approx 0.15A.$$
$$V_{canal5} \approx 1.05V \rightarrow V_{ventilador} = ((33 + 3.3) * 12V) / 3.3 \approx 11.5V$$

La tensión en el motor no alcanza los 12V debido a las distintas caídas de tensión en el diodo de entrada y en la resistencia shunt.

5.3.5.2. Control final de motor DC:

Una vez comprobado el funcionamiento del código generado y de la placa de control construida, se procedió a realizar el montaje final del sistema de control del motor DC de 24V disponible en los laboratorios del departamento de Ingeniería Eléctrica (figura 5.9). Para esta tarea es necesario disponer de dos fuentes de alimentación de 24V de continua: una para alimentar la excitación del motor y otra para alimentar al mismo a través de la placa de control. Por otro lado se emplea un osciloscopio para la monitorización de la señal de PWM control generada por el DSP, además de ser útil para la realización de determinadas medidas. También se utilizó un tacómetro para la medida de la velocidad alcanzada por el motor. Para realizar los experimentos con carga, se dispuso en la misma bancada del motor una carga de 24V que sería alimentada mediante una fuente de tensión de 15V de continua. Una vez realizadas las conexiones de los distintos elementos y del DSP con el ordenador el sistema queda como se muestra en la figura 5.21.

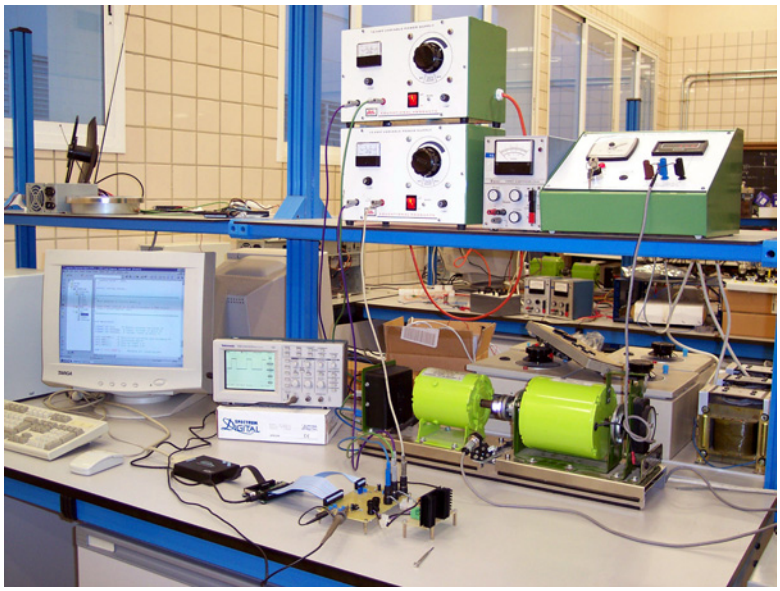


Figura 5.21. Sistema completo de control de motor DC.

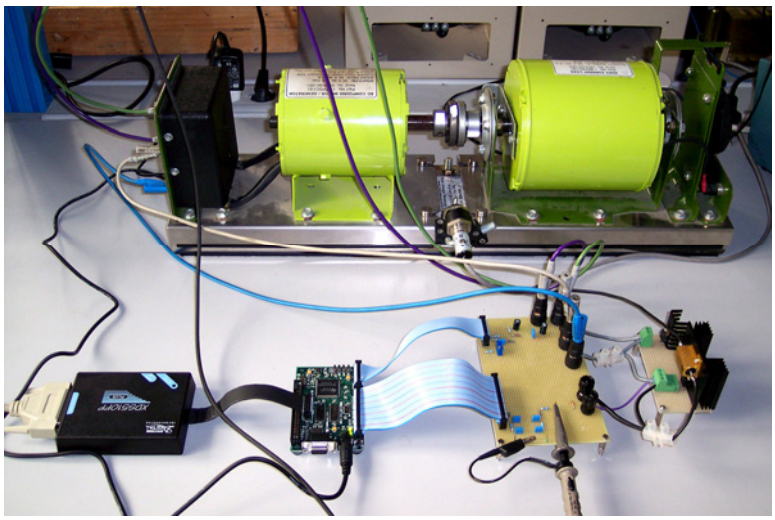


Figura 5.22. Motor DC con placa de control.

El conexionado del sistema se realiza según las indicaciones dadas por los esquemas de las figuras 5.11, 5.12 y 5.22. En ésta última se observa cómo se realizan las conexiones del motor a la placa de control y ésta al DSP.

Una vez realizado el montaje, su puesta en marcha debe realizarse según los pasos detallados en el apartado anterior para evitar daños a los dispositivos. Para comprobar el correcto funcionamiento del sistema se hace necesaria la utilización de equipos de medida, como son osciloscopio, voltímetro o amperímetro. Para la monitorización de la señal PWM de control generada por el DSP se conecta una sonda de osciloscopio a la placa de control en el pin habilitado para ello, ya sea para medir la onda filtrada o sin filtrar (figura 5.16). En el osciloscopio pueden realizarse diferentes mediciones acerca de la misma señal, como son su frecuencia, valor medio, valor RMS, etc. (figura 5.23). También permite observa en tiempo real cómo se modifica el ancho del pulso PWM y su valor medio cuando se manipula el potenciómetro de control de velocidad, dando información sobre la tensión que se está alimentando al motor.

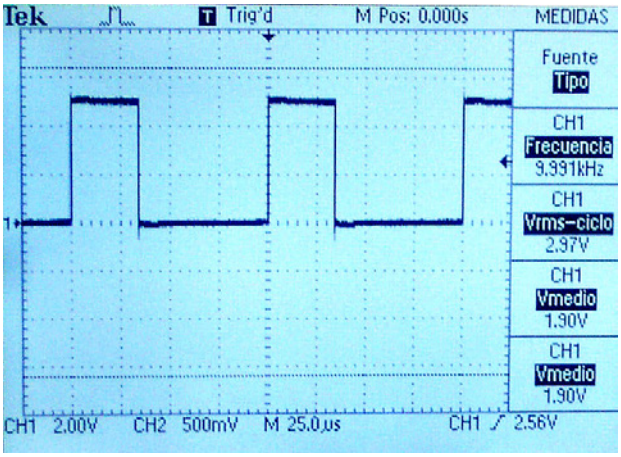


Figura 5.23. Señal PWM de control en osciloscopio.

Un parámetro claramente importante en un sistema de control de velocidad de un motor es precisamente la velocidad que alcanza el mismo. Para ello, en el sistema es necesario disponer de un tacómetro colocado en la bancada del motor y apuntando a su eje, además de disponer de su correspondiente marcador de velocidad (figura 5.24). En él se muestra en tiempo real la velocidad que tiene el motor en cada instante del proceso de control, comprobándose la eficiencia del control generado por el DSP.

Las pruebas realizadas se han llevado a cabo con el código detallado con anterioridad, sin realizar ningún cambio en los factores de escala como ocurría con el ventilador. En ellas principalmente se realizaron pruebas sin carga en el motor para comprobar la velocidad máxima alcanzada por el mismo además de la intensidad que circula por el circuito. Los resultados fueron los siguientes:

Vacío: $V_{Alimentación} = 24V$ $V_{Motor} = 22.9V$ $I_{Motor} = 1.34A$ Velocidad = 1450 rpm

La tensión en el motor no llega a los 24V debido a las caídas de tensión que se producen en otros componentes del circuito.

Figura 5.24. Equipo de medida de velocidad del motor.

El factor de escala empleado para la medida de la potencia absorbida está calculado para representar como máximo 50W (24V * 2.1A), de manera que 5V en el osciloscopio equivalen a esos 50W de potencia absorbida, es decir, 1V en el osciloscopio representa 10W de potencia. Para comprobarse la precisión de la medida de potencia por parte del DSP se modificó la velocidad del motor hasta que la intensidad de circulación fuese de 1A. Al ser la tensión de alimentación de 24V, la potencia del circuito en ese instante era de P = 1A*24V = 24W. En esa situación la potencia medida por el DSP oscilaba entorno a 2.4V en el osciloscopio, con lo que la precisión es bastante correcta. El factor de escala puede modificarse para representar hasta los 125W de potencia que puede consumir el motor. Para ello basta con emplear la fórmula detallada anteriormente.

Se realizaron pruebas en las que se comprobó hasta que velocidad mínima podía controlarse al DSP. Después de varios intentos se consiguió dejar al motor a una velocidad de 30 rpm, momento en el cual la intensidad que circula por el mismo es de 0.42A. En esta situación también se realizaron pruebas a la precisión de la medida de potencia del DSP, que para este caso oscilaba entorno a 1V, lo cual concuerda bastante con la potencia absorbida en ese momento: P = 0.42A*24V = 10.08W.

Las últimas pruebas que se realizaron con el motor fueron añadiéndole una carga conectada a su eje. Esta carga era alimentada por una fuente variable de tensión continua de 15V, de manera que a más tensión aplicada, más carga se aplicaba al motor. El procedimiento seguido fue establecer el motor en vacío a su velocidad máxima, para luego comenzar a aplicarle tensión a la carga. En el momento de la redacción de este proyecto, los resultados obtenidos en carga no fueron satisfactorios. Esto se debe a que al aplicarle carga al motor la intensidad de circulación aumentaba como se esperaba, pero la tensión que alimentaba al motor descendía, con lo que también lo hacia su velocidad. En pruebas realizadas se fue aumentando la tensión de alimentación de la carga hasta unos 9V, momento en el que la intensidad de circulación era de 2.2A, pero la tensión que llegaba al motor había descendido por debajo de los 15V, quedando este casi detenido a unas 200 rpm. En ese punto se optó por no aplicarle más carga dado que

el transistor comenzaba a calentarse en exceso. Esta anomalía puede deberse a que se produzcan caídas de tensión inesperadas en algunos de los componentes del circuito, de manera deje de llegarle toda la tensión de alimentación al motor. En particular puede que estas caídas se produzcan en el transistor o en los diodos debido a la variación de su resistencia con el aumento de temperatura. De todas formas, se aprovecharon esos valores de tensión y de intensidad para comprobar una vez más la precisión de la medida de potencia realizada por el DSP, la cual debía ser de P = 2.2A*24V = 52.8W. En efecto el osciloscopio marcaba un valor de 5V al estar el máximo de potencia establecido en 50W por el factor de escala aplicado.

5.3.6. Mejoras del circuito:

La aplicación aquí descrita es una adaptación de un tutorial desarrollado por Texas Instruments para el procesador TMS320LF2407, el cual permite realizar múltiples mediciones con facilidad, con lo que hubo de adaptarse el código y los elementos empleados en el circuito para su implementación con el motor DC de 24V. Como es conocido, más que la potencia absorbida por el sistema completo, lo que realmente interesante es la potencia absorbida por el propio motor. Este punto no pudo llevarse a cabo satisfactoriamente debido a los problemas encontrados a la hora de adaptar el código para realizar más de dos medidas con el convertidor analógico-digital del TMS320F243. Estas limitaciones del procesador hicieron que en un principio el diseño de la placa desarrollada fuese el comentado anteriormente, ya que se carecía de documentación en la que se detallara cómo se podían realizar varias mediciones. Finalizando el proyecto se lograron implementar más de dos medidas con el método descrito en el código empleado, aunque posiblemente no sea el algoritmo más eficiente para lograrlo. La consecución de este objetivo da pié a intentar medir la tensión que alimenta al motor y, por tanto, calcular la potencia que consume. Esta mejora del sistema podría realizarse midiendo la tensión en las dos bornas de conexión del motor (puntos A y C de figura 5.25), cuya diferencia sería la tensión que lo alimenta.

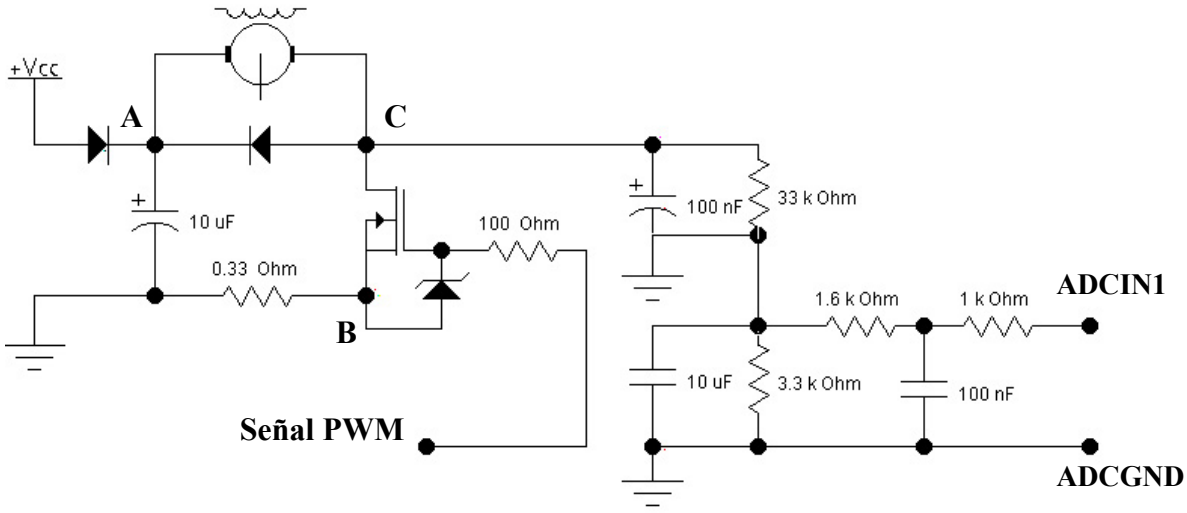


Figura 5.25. Posible modificación del circuito.

Dado que la medida de tensión en una de las bornas ya se encuentra implementada en el circuito mediante un divisor de tensión en el punto A (figuras 5.11 y 5.25) y es medida en el canal ADCIN5, únicamente queda modificar el circuito para medir en otro canal (por ejemplo el ADCIN1) la tensión de la otra borna de conexión (punto C de figuras 5.11 y 5.25). Esto podría realizarse empleando un divisor de tensión similar al ya empleado en el punto A. La tensión en esta borna depende del estado de conducción y corte del transistor, por lo que será una señal cuadrada que oscile entre 24V y 1V aproximadamente (tensión de la resistencia shunt). Es por ello que es necesaria la conexión de un filtro RC que permita al convertidor leer el valor medio de la tensión en ese punto. Este filtro RC puede ser similar al empleado para la medida de la intensidad del motor mediante la tensión de la resistencia shunt (punto B de figuras 5.11 y 5.25).

La aplicación de esta modificación (figura 5.25) o de otras que pudieran producirse permitirían la medición de la tensión que es aplicada al motor, por lo que se podría efectuar el cálculo de la potencia consumida por el mismo. Para ello posiblemente bastase con modificar una pequeña parte del código empleado hasta ahora. En particular se trataría de modificar la función "measure_voltage" encargada hasta ahora de medir la tensión de alimentación al sistema. En ella habría de incluirse también el cambio del registro de control ADCTRL1 para medir el nuevo canal mediante el convertidor ADC1, el cual guarda el valor medido en ADCFIFO1. De esta forma, ambas FIFOs deberían ser vaciadas para eliminar los antiguos valores pertenecientes al anterior canal configurado. El vaciado de las FIFOs se realiza rápidamente mediante la carga en el registro acumulador del primer valor de la misma. Esto se consigue mediante la instrucción "LACL 7036h" o "LACL 7038h", siendo 7063h la dirección de ADCFIFO1 y 7038h la dirección de ADCFIFO2. Este método es útil y rápido ya que esos valores no van a ser empleados, por lo que no merece la pena guardarlos en variables, aunque también podría realizarse así. Una vez realizado el cambio se procedería a la medición de ambos canales, cuyos resultados son restados para obtener la tensión aplicada al motor. Después de escalar este valor de manera pertinente, se devuelve el registro ADCTRL1 al estado inicial. Por último, la función devolvería al programa principal la tensión hallada para el cálculo de la potencia consumida por el motor. Con todo ello la función modificada quedaría como sigue:

```
volatile signed int measure_voltage(void)
{

 signed int return_value;              /* Definición de variables empleadas */
 signed long scaled;
 signed long scaled1;
 signed long scaled2;
 signed int x;
 x=0;

/* Cambio de la configuración del registro ADCTRL1 para leer los canales 1 y 5 */

 ADCTRL1=((ADCTRL1 & 0xFF81) | (0x0005<<4) | (0x0001<<1);

 ADCTRL1|=0x0100;                      /* Borra flag de interrupción pendiente */
```

```
 asm (" LACL 7036h");              /* Vaciado del la pila ADCFIFO1 para eliminar
 asm (" LACL 7036h");                    valores almacenados de otros canales */
 asm (" LACL 7036h");

 asm (" LACL 7038h");              /* Vaciado del la pila ADCFIFO2 para eliminar
 asm (" LACL 7038h");                    valores almacenados de otros canales */
 asm (" LACL 7038h");

 while((ADCTRL1 & 0x0100)==0)      /* Bucle de espera mientras se genera
 x++;                                    una nueva interrupción */

 x=0;

 while(ADCTRL1 & 0x0080)           /* Bucle de espera mientras finaliza
 x++;                                    la medida en el canal 5 */

 scaled1 = (signed long)(ADCFIFO1 >> 1);      /* Hace sitio al bit de signo */

 scaled2 = (signed long)(ADCFIFO2 >> 1);      /* Hace sitio al bit de signo */

 scaled = scaled2- scaled1;

 scaled *= 0x2B02;    /* Multiplica por factor de escala para dar valor entre  0 y 5500 */


/* Vuelta al estado inicial del registro ADCTRL1 para leer canales 0 y 3 */

 ADCTRL1=((ADCTRL1 & 0xFF81) | (0x0003<<4));

 ADCTRL1|=0x0100;                  /* Borra flag de interrupción pendiente */

 asm (" LACL 7036h");              /* Vaciado del la pila ADCFIFO1 para eliminar
 asm (" LACL 7036h");                    valores almacenados de otros canales */
 asm (" LACL 7036h");

 asm (" LACL 7038h");              /* Lecturas de ADCFIFO2 para eliminar
 asm (" LACL 7038h");                    valores almacenados de otros canales */
 asm (" LACL 7038h");

 return_value = (signed int)(scaled >> 16);

 return(return_value);             /* Devuelve valor escalado */

}
```

Otras posibles mejoras al circuito y la aplicación sería la incorporación de señales procedentes de un encoder para la monitorización de la velocidad del motor por el DSP. Ello requeriría de la introducción de nuevos componentes en la placa que adapten las señales para su conexión a las entradas de captura del DSP, además de la modificación del código empleado para implementar un mejor control del motor.

5.3.7. Aplicación con VisSim:

Como se ha comentado con anterioridad, el software de desarrollo VisSim permite la generación de aplicaciones mediante la unión de bloques funcionales, a partir de los cuales generar un código para su ejecución en un DSP. Esta facilidad para generar aplicaciones puede ser empleada también para implementar la aplicación descrita anteriormente de manera mucho más sencilla para usuarios con poca experiencia en lenguaje C. Además, al tratarse de un programa desarrollado en conjunción con Texas Instruments, es capaz de generar aplicaciones en las que se requieran la medición de varios canales del convertidor ADC. Esto último es especialmente importante, ya que nos permite generar una aplicación eficiente en la que se miden múltiples canales analógicos.

Al igual que este programa presenta una serie de ventajas, también presenta una serie de inconvenientes. Estos además, limitan bastante el diseño y comprobación de los diagramas generados. En capítulos anteriores se han detallado las distintas limitaciones existentes en determinados bloques funcionales, como es el bloque de generación PWM. Este bloque únicamente podía ser empleado en modo interactivo empleando una configuración fija determinada por VisSim: Temporizador 1, Prescaler 16, Periodo 12500 y modo de cuenta Up. Cualquier otra configuración no funciona en modo interactivo. A su vez, existe el impedimento de emplear el temporizador 2 en modo interactivo. Únicamente puede emplearse en modo compilado. De todas formas el fabricante recomienda emplear los módulos PWM siempre en modo compilado. Todo esto hace que en modo interactivo sólo pueda generarse una única onda PWM y de sólo 500Hz. Esta baja frecuencia impide que pueda ser empleada en el control del motor DC de 24V, ya que provoca la generación de una gran cantidad de ruido en el motor que molesta al usuario. Por el contrario si es suficiente para controlar el ventilador de 12V sin problemas en modo interactivo. Como consecuencia de todo esto, a la hora de generar una aplicación para el motor de 24V siempre es necesario compilar parte del diagrama y descargarlo al DSP para poder ponerlo a prueba. Esto no sería un problema grave si no fuese porque, hasta el momento de la redacción de este proyecto, en el PC empleado para las pruebas en el laboratorio, las numerosas versiones de VisSim empleadas dan múltiples errores a la hora de la compilación, fallando por completo la generación del archivo ejecutable. Ello ha evitado el que puedan realizarse experimentos en los que se pudiera emplear el motor DC de 24V. Curiosamente este problema en la compilación no se presenta en otros ordenadores particulares en los que se realizaron distintas pruebas. Se intentó compilar los diagramas en uno de estos PC´s y llevar el archivo generado ".out" al PC del laboratorio, pero todas las pruebas condujeron a errores del VisSim. Este problema se puso en conocimiento del distribuidor en España de VisSim y hasta la fecha no hay solución al problema. Debido a todos estos problemas, las anotaciones que siguen a continuación debieron realizarse empleando únicamente el ventilador de 12V.

Para desarrollar un diagrama que realice las mismas funciones que el código anterior, basta con situar los bloques funcionales que sean necesarios (entradas al convertidor ADC y señales PWM) e interconectarlos de manera adecuada. Para hacerlos más entendible a otros usuarios, pueden incluirse displays y representaciones gráficas que muestren los valores de las distintas magnitudes monitorizadas en cada momento. Un ejemplo de cómo podría realizarse la aplicación se muestra en la figura 5.26.

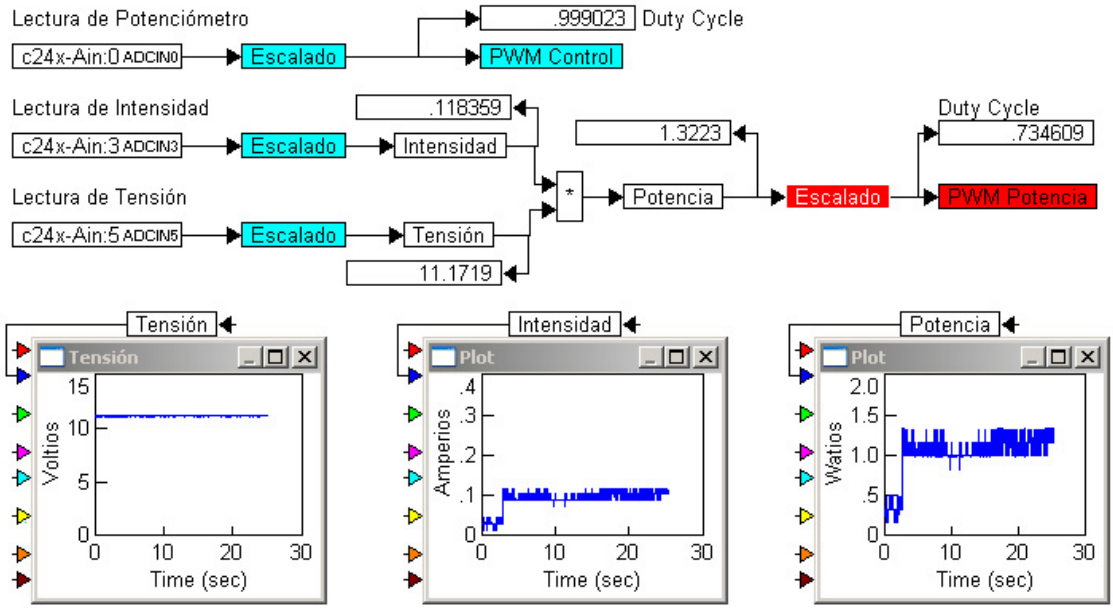


Figura 5.26. Aplicación desarrollada con VisSim.

En este ejemplo se procede a la lectura de los tres canales analógicos ADCIN0, ADCIN3 y ADCIN5. Como VisSim está preparado para que estos bloques den como resultado un valor entre 0 y 5V, para representar las distintas magnitudes deben ser escalados convenientemente. Esto se realiza en el diagrama mediante los bloques denominados "Escalado", cada uno de los cuales convierte la medida anterior en un valor en la escala de la variable correspondiente. Por ejemplo, como la tensión oscila entre 0 y 55V por la constitución de la placa, su escalado se realizará dividiendo inicialmente entre 5 para luego multiplicarse por 55, obteniéndose así un número en la escala de 0 a 55V (figura 5.27).

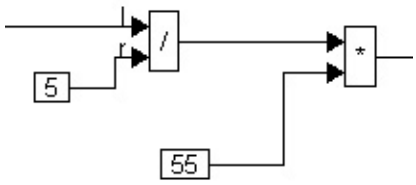


Figura 5.27. Escalado de la tensión.

Se procede de igual manera para el escalado de la intensidad, que oscila en un rango de 0 a 15.15A. Para el potenciómetro únicamente se divide por 5, ya que sólo se necesita un valor para el Duty Cycle de la señal PWM entre 0 y 1. Recordar que para poder generar las señales PWM el Duty Cycle debe convertirse previamente en un número entero de coma fija 1:16.

Una vez escaladas las magnitudes se procede al cálculo de la potencia y del Duty Cycle de la onda PWM que la representa. Para ello basta con multiplicar la tensión e intensidad recientemente escaladas. Como este desarrollo se ha basado en el uso del ventilador de 12V, la potencia máxima que absorberá el circuito será P = 0.15A*12V = 1.8W. Por tanto el escalado se realiza dividiendo por 1.8 para obtener un valor del Duty Cycle entre 0 y 1.

Para no complicar el diagrama con líneas, se ha empleado el uso de variables en VisSim para llevar los valores de las magnitudes a su correspondiente representación gráfica. También se han dispuesto displays junto a las variables para comprobar su valor en cada instante.

Para la generación de las ondas PWM es necesario compilar el diagrama, ya sea completamente, o compilando solamente los bloques PWM después de agruparlos en un solo bloque compuesto (figura 5.28). Esto se debe a que el PWM generado con el temporizador 2 no funciona en modo compilado bajo ninguna configuración. Al tener que compilar obligatoriamente, será necesario emplear una Interfaz DSP para incluir el archivo generado en el resto del diagrama (figura 5.29).

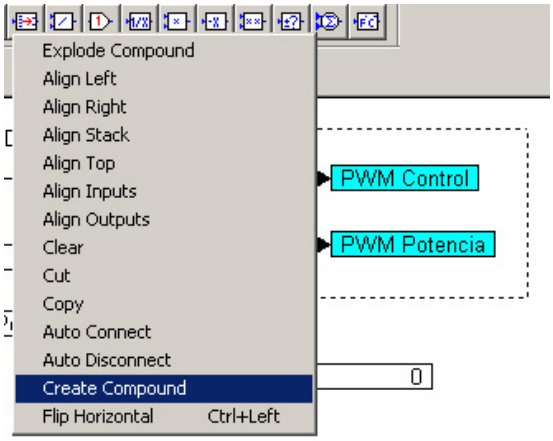


Figura 5.28. Creación de bloque PWM compuesto.

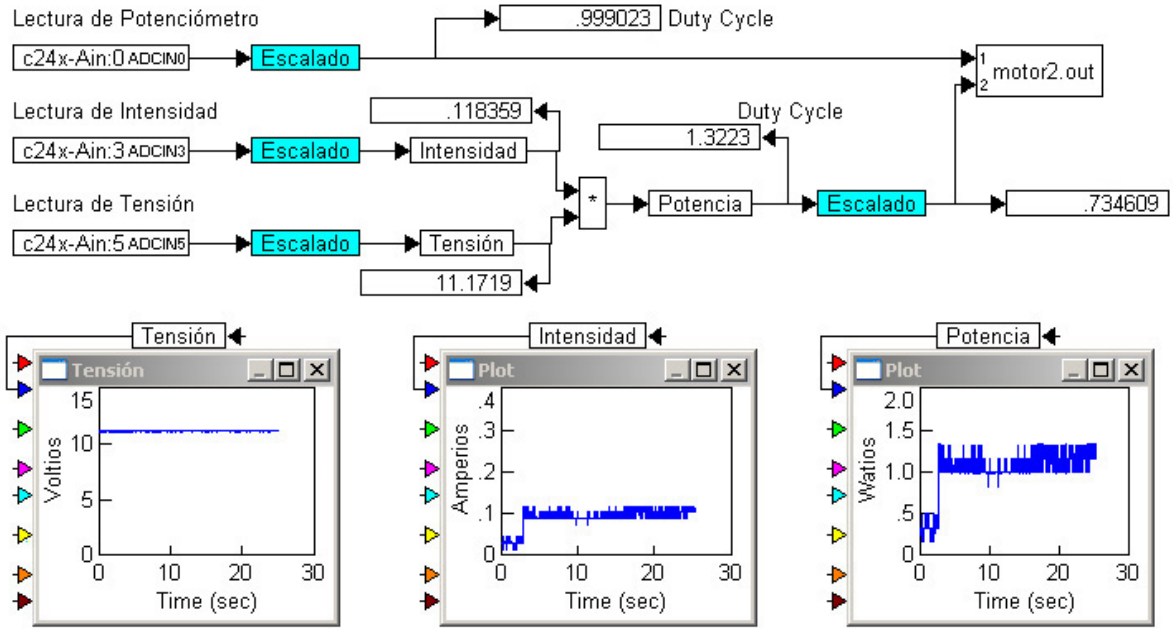


Figura 5.29. Diagrama con Interfaz DSP.

El hecho de compilar las señales PWM permite que estas se puedan configurar con los parámetros que el usuario crea necesarios, como por ejemplo que tengan 10kHz de frecuencia. Esto último se llevaría a cabo con la configuración de la figura 5.30.

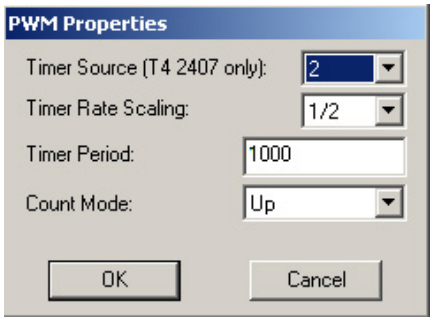


Figura 5.30. Configuración de PWM a 10kHz.

En las figuras 5.26 y 5.29 se aprecia bastante ruido en la señal de potencia. Esto se debe al ruido que introduce la medida de la intensidad del ventilador por los valores bastante pequeños que toma, lo que impide que sean medidos con precisión si existe ruido en el entorno de trabajo. Este ruido se ve amplificado al ser multiplicado por la tensión para hallar la potencia.

Otra variante de este diagrama puede ser la sustitución del canal de lectura del potenciómetro por un deslizador, de forma que la velocidad pase a controlarse desde el PC. Este deslizador puede variar en una escala que represente la velocidad del motor, para lo cual es necesario dividir por la velocidad máxima del motor en el escalado para obtener un Duty Cycle entre 0 y 1.

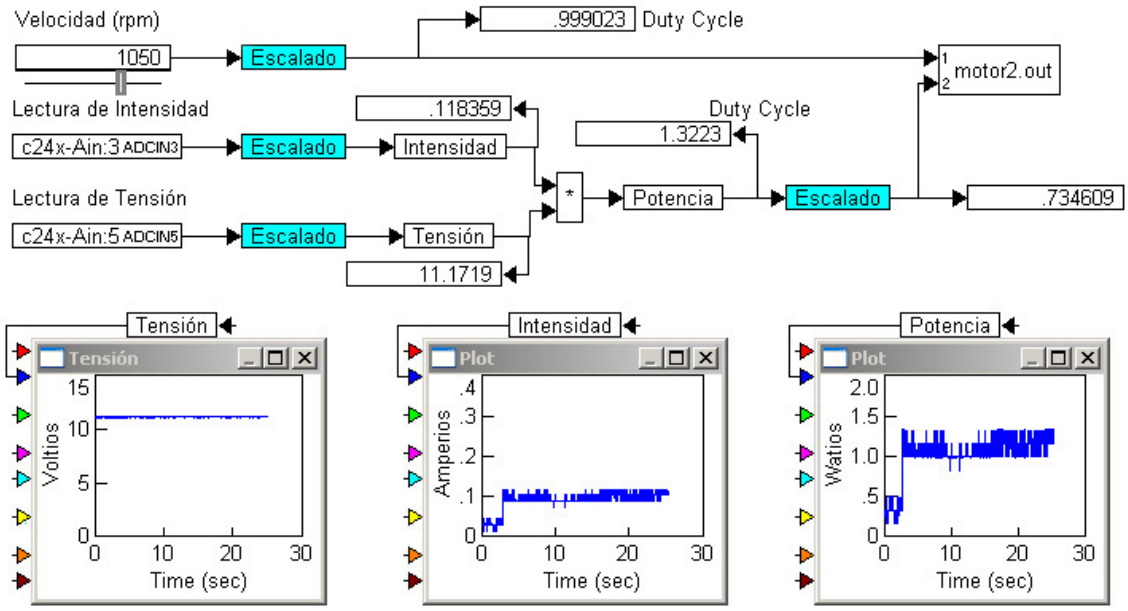


Figura 5.31. Control de velocidad desde el PC.

Una última modificación que podría realizarse es la estimación de la velocidad que tiene el motor en cada instante. Una forma sencilla de hacerlo, pero no muy exacta, es mediante una estimación lineal a través del cálculo de la potencia, es decir, a plena potencia se supone que la velocidad es máxima, con lo que cuando se consuma la mitad de esa potencia el motor estará a la mitad de su velocidad máxima. De esta manera, calculando la potencia, dividiéndola entre la potencia máxima y multiplicando el resultado por la velocidad máxima del motor, se obtendría una estimación de la velocidad en ese instante (figura 5.33). Para evitar excesivos errores en las operaciones es mejor realizar la multiplicación antes de la división, de forma que se dividan valores no demasiado pequeños (figura 5.32).

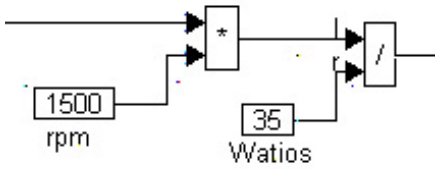


Figura 5.32. Estimación velocidad.

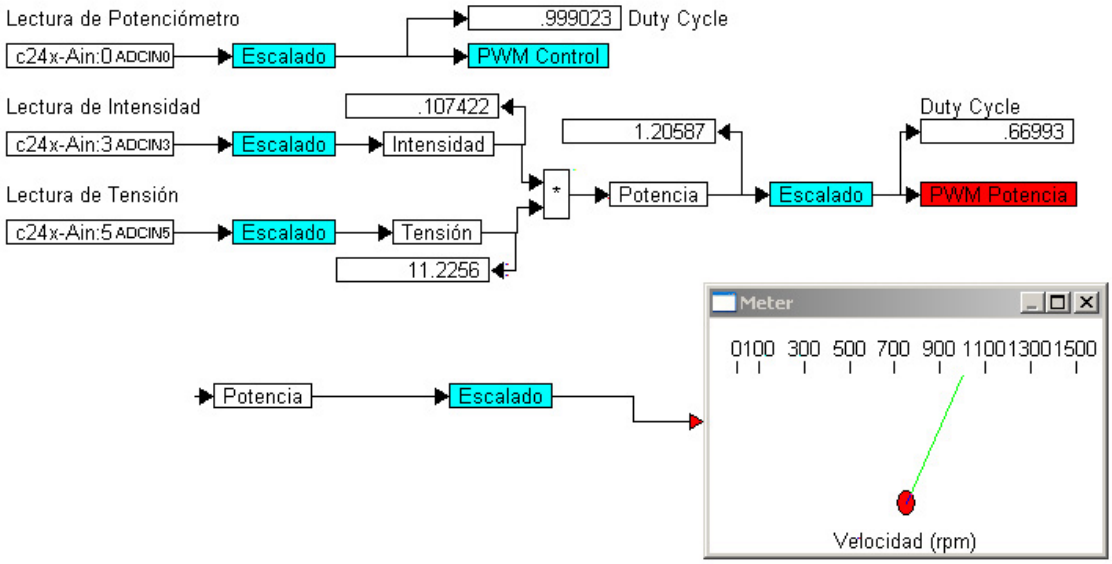


Figura 5.33. Monitorización de velocidad con VisSim.

Para monitorizar la velocidad puede emplearse un display o, como se muestra en la figura 5.33, un medidor gráfico a modo de velocímetro, el cual se encuentra entre los iconos de acceso rápido de la barra superior de menús entre los iconos de representación gráfica.

APLICACIONES Y LÍNEAS FUTURAS

El objetivo de este documento pretende establecer los cimientos que permitan desarrollar aplicaciones más complejas de control de motores basadas en DSP´s, ya que forma parte de un proyecto común más amplio para desarrollar un banco de pruebas en los laboratorios del departamento de Ingeniería Eléctrica que simule la variación de carga que sufre un vehículo eléctrico dependiendo del terreno por el que circule. Para ello se han analizado las distintas posibilidades que presenta el DSP, como son sus periféricos y software de desarrollo, desarrollando códigos de ejemplo para una mejor comprensión de cada uno de ellos por parte del lector. De esta manera el presente documento intenta ser un manual para los no iniciados en el uso de DSP´s, de forma que tengan un punto de partida del cual empezar a desarrollar aplicaciones más complejas y eficientes. Además, este proyecto ha servido para desarrollar una serie de placas de prueba y aplicaciones que permiten poner en práctica físicamente los conocimientos analizados a lo largo del documento, evitando dejar en el aire la demostración práctica de los mismos. Así, el usuario que no haya manejado nunca procesadores de este tipo puede poner en práctica todo lo aquí detallado, lo que le sirve como punto de partida práctico y teórico para la investigación y desarrollo de nuevas aplicaciones.

Respecto al control de motores, en este documento se ha analizado una sencilla aplicación de control de un motor de continua, el cual es susceptible de ser mejorado ampliamente. En el apartado 5.3.6 del capítulo anterior se detallaban algunas posibles mejoras a la aplicación desarrollada, las cuales no pudieron llevarse a cabo de forma experimental debido a la indisposición de tiempo y material necesario. Por ello, una ampliación inmediata de la aplicación documentada es la inclusión en el circuito y en el código de todo lo necesario para proceder a medir la tensión que llega a las bornas del motor, de forma que pueda calcularse la potencia consumida por el mismo, ya que esto resulta más interesante que la consumida por el sistema empleada hasta el momento. Para ello pueden emplearse las sugerencias establecidas en el apartado 5.3.6 del capítulo anterior.

Una ampliación muy interesante que podría llevarse a cabo es el uso de un encoder de pulsos en cuadratura, de manera que el DSP proceda a medir la velocidad del motor en cada momento, pudiéndose implementar así un algoritmo de control mucho más eficaz que el desarrollado en este documento. Para ello puede ser necesario la modificación de la placa de control diseñada para incluir nuevos componentes que adapten las señales al nivel que emplean las señales de captura del DSP. Incluyendo este dispositivo se cierra el bucle del control, permitiendo que puedan ser desarrolladas aplicación que implementen distintos algoritmos de control de velocidad: control difuso, PI, PID, etc. Ello puede llevarse acabo con la ayuda de las distintas funciones ya desarrolladas por Texas Instruments en su Librería de Control Digital. En ella se definen multitud de funciones en lenguaje C empleadas en algoritmos de control digital de motores, ya que se encuentran optimizadas por el propio fabricante de DSP´s. Con la ayuda de estas herramientas puede darse el salto al control de motores de inducción y de alterna, puesto que son empleados ampliamente en multitud a aplicaciones. Para conseguir este objetivo Texas Instruments pone a disposición del usuario gran variedad de documentos en su página web en los que se detalla cómo pueden ponerse en práctica el control de distintos tipos de motores: inducción trifásico, brushless, paso a paso, de reluctancia conmutada, etc.

Otra posible vía de investigación puede ser el uso del software de desarrollo VisSim para realizar aplicaciones de control más complejas, ya que dispone de numerosos bloques funcionales que permiten realizar distintos tipos de control, además de poder simular en el PC el control sobre diferentes tipos de motores. Esto no pudo realizarse a lo largo del presente proyecto debido, en parte, a los numerosos problemas que surgieron empleándose dicho programa en el ordenador dispuesto en el laboratorio para el desarrollo del proyecto. Algunas de las complicaciones que presenta este programa son debidas a la falta de comprobación del correcto funcionamiento del mismo por parte del fabricante, por lo que algunos bloques no funcionan adecuadamente. En capítulos anteriores se comentaron cuales de estos bloques se ven afectados, y que el fabricante del programa intenta ir corrigiendo estos errores a medida que se les van notificando por correo electrónico, sacando nuevas versiones revisadas del programa en su página web cada cierto tiempo.

En la teoría, el uso de este programa simplifica en gran medida la programación de aplicaciones de control de motores en un DSP, ya que dispone de todas las herramientas necesarias en forma de bloques funcionales fácilmente configurables. De esta manera pueden implementarse algoritmos de control PI simplemente colocando dicho bloque en el diagrama desarrollado. Para el diseño del control de velocidad de un motor el programa dispone de bloques funcionales que permiten obtener la velocidad del motor en cada instante, con lo que pueden llegar a desarrollarse algoritmos de control muy eficientes.

La posibilidad de ejecutar los diagramas en modo interactivo permiten que puedan ser comprobados antes de la generación del archivo ejecutable definitivo. Además dispone de la posibilidad de monitorización de algunas señales en tiempo real, pudiéndose así diseñar una interfaz gráfica más o menos sencilla en la que representar gráficamente algunas magnitudes, además de introducir fácilmente los valores de ciertos parámetros de configuración de la aplicación. Todo esto hace que VisSim sea una herramienta a tener muy en cuenta para las posibles líneas futuras de investigación con DSP´s.

BIBLIOGRAFÍA

CAPÍTULO 1: INTRODUCCIÓN A LOS DSP´S

- Kenneth W. Schachter. *An intuitive approach to Digital Signal Processing*. Texas Instruments.

- *A smarter, faster way for embedded control design*. VisSim Embedded Controls Developer, Visual Solutions.

- *Controlador DSP C240*. Eurofach Electrónica, Septiembre 1996.

- *Microprocessor for Power Electronics and Electrical Drives Applications*. IEEE Industrial Electronics Society Newsletter, September 1999.

- *C24x DSP controller*. Details on signal processing, ISSUE 45, Texas Instruments, September 1996.

- *TMS320C24x DSP´S advance the design of motor-control applications*. Details in digital control system, ISSUE 1, Texas Instruments, July 1998.

- *TMS320F240: DSP power for digital motor control applications*. New Products Showcase, ISSUE 28, Texas Instruments, May 1997.

- *TMS320C24x DSP Controller*. Product Bulletin, Texas Instruments.

- *New DSP controllers deliver twice the MIPS for the money*. Details on signal processing, ISSUE 55, Texas Instruments, July/August 1999.

- *TMS320C24x DSP´S: Optimized for Motor Control*. Texas Instruments, 1999.

- Sergio Lorenzi. *DSP Nuevas arquitecturas y mayor potencia de cálculo*. Dossier, Mundo Electrónico, Nº 308, Abril 2000.

- Jordi Salazar. *Procesadores digitales de señal (DSP). Arquitecturas y criterios de selección*. Perspectiva, Mundo Electrónico, Nº 314, Noviembre 2000.

- Juan José Salgado. *Software de simulación y cálculo en ingeniería*. Dossier, Mundo Electrónico, Nº 315, Diciembre 2000.

- Juan José Salgado. *Software para procesos en tiempo real. Herramientas de modelización de sistemas.* Dossier, Mundo Electrónico, Nº 311, Julio 2000.

- Juan José Salgado. *Emuladores. Unas herramientas imprescindibles*. Dossier, Mundo Electrónico, Nº 309, Mayo 2000.

CAPÍTULO 2: KIT DE INICIACIÓN DSK F243

- *F24x DSK Setup and Tutorial.* Spectrum Digital, 1999.

- *TMS320F24X DSP Starter Kit (DSK) Setup Instructions.* Spectrum Digital, 2000.

- *EVM320F243 System Kit C2000 Code Composer 4.XX Development Tools Setup Instructions*, Spectrum Digital, 2000.

- *eZdspTM TMS320LF2407 Hardware Kit C2000 Code Composer 4.XX Development Tools Setup Instructions*, Spectrum Digital, 2001.

- *XDS510PP PLUS Parallel Port JTAG Emulator. Installation Guide*. Spectrum Digital, 2002.

- *XDS510PP PLUS JTAG Emulator Code Composer/Studio Development Tools Setup Instructions.* Spectrum Digital, 2001.

- *Emulation Fundamentals for TI´s DSP Solutions*. Application Report SPRA439B. Texas Instruments, May 2002.

- *F243 DSK Technical Reference.* Spectrum Digital, 1999.

- *TMS320F243 Evaluation Module Technical Reference.* Spectrum Digital, 2000.

- *eZdspLF2407 Technical Reference.* Spectrum Digital, 2001.

- *TMS320F243, TMS320F241 DSP Controllers*. SPRS064C, Texas Instruments, December 1997, Revised September 2000.

- *TMS320F20x/F24x Embedded Flash Memory Technical Reference.* SPRU282. Texas Instruments, September 1998.

- *TMS320F/C24x DSP Controllers Reference Guide, Peripheral Library and Specific Drivers*. SPRU161C. Texas Instruments, June 1999.

- *TMS320LF/LC240xA DSP Controllers Reference Guide. System and Peripherals*. SPRU357B. Texas Instruments, December 2001.

- J. Cerdá, V. Herrero, F. Ballester y A. Sebastiá. *Sistemas de control dedicados. Aplicación del bus CAN al control de ascensores.* Perspectiva, Mundo Electrónico, Nº 314, Noviembre 2000.

CAPÍTULO 3: SOFTWARE DE DESARROLLO DE APLICACIONES

- *TMS320F/C24x DSP Controllers Reference Guide. CPU and Instructions Set*. SPRU160C, Texas Instruments, June 1999.

- *Getting Started in C and Assembly Code With the TMS320LF240x DSP*. Application Report SPRA755A. Texas Instruments, July 2002.

- *C2000 Teaching Materials CD-ROM. Tutorials and Applications*. Texas Instruments.

- *Spectrum Digital Symbolic Assembler for the F24x DSP Technical Reference*. Spectrum Digital, July 1999.

- *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools. User's Guide*. SPRU018D, Texas Instruments, 1995.

- *TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide.* SPRU024E. Texas Instruments, August 1999.

- *Code Composer Quick Start Guide*. Texas Instruments, 1999.

- *Code Composer User's Guide*. SPRU296A. Texas Instruments, October 1999.

- *TMS320C2xx/C24x Code Composer User´s Guide*. SPRU490. Texas Instruments, October 2000.

- *JTAG FAQ.* Spectrum Digital, September 2001.

- *Emulator FAQ*. Spectrum Digital, September 2002.

- *Product FAQ*. Spectrum Digital, September 2002.

- *Software FAQ*. Spectrum Digital, September 2002.

- *VisSim User´s Guide.* Visual Solutions.

- *VisSim Embedded Controls Developer User´s Guide.* Visual Solutions.

- Federico José Barrero García. *Complementos de sistemas electrónicos digitales*. Apuntes de Cátedra.

CAPÍTULO 4: REGISTROS Y PERIFÉRICOS DEL TMS320F243

- *TMS320F243/F241/C242 DSP Controllers. System and Peripherals*. SPRU276C. Texas Instruments, January 2000.

- Fco. J. Gimeno Sales, Salvador Seguí Chilet, *Procesador Digital de señal DSP: TMS320LF240x. Arquitectura y Aplicaciones.* Editorial Universidad Politécnica de Valencia 2003.

- Hamid A. Toliyat, Steven Campell. Texas A&M University. Department of Electrical Engineering, College Station, Texas. *DSP-Based Electromechanical Motion Control.* CRC PRESS 2004.

- *Web: Texas Instruments DSP Laboratory. 'Hands on TMS320F243/LF2407'*. Frank Bormann. University of Applied Sciences Zwickau. Germany.

- *16 bits DSP Microcontroller Texas Instruments TMS320LF2407. Module: Examples EVMLF2407 Kit*. Frank Bormann. University of Applied Sciences Zwickau. Germany.

- *TMS320C24x General Purpose Timer 1 Asymmetric Mode*. Spra367. Texas Instruments 1996.

- *TMS320C24x General Purpose Timer 1 symmetric Mode*. Spra368. Texas Instruments 1997.

- *TMS320C24x PWM Full Compare in symmetric Mode*. Spra369. Texas Instruments 1997.

- *TMS320C24x PWM Simple Compare in symmetric Mode*. Spra370. Texas Instruments 1997.

- *Creating a sine modulated PWM signal using the TMS320F240 EVM.* Spra411. Texas Instruments. January 1999.

- *Using PWM output as an digital to analog converter on a TMS320C240 DSP.* Spra490. Texas Instruments. November 1998.

CAPÍTULO 5: APLICACIÓN DE LOS DSP´S AL CONTROL DE MOTORES

- Fco. J. Gimeno Sales, Salvador Seguí Chilet, *Procesador Digital de señal DSP: TMS320LF240x. Arquitectura y Aplicaciones.* Editorial Universidad Politécnica de Valencia 2003.

- *Creating a pulse width modulated signal with a fixed duty cycle using the TMS320F240 EVM.* Spra410. Texas Instruments. January 1999.

- *Generating a PWM signal modulated by an analog input using the TMS320F240 EVM.* Texas Instruments. January 1999.

- *TMS320F240 DSP Solution for Obtaining Resolver Angular Position and Speed.* Spra605. Texas instruments. February 2000.

- *Using the Capture Units for Low Speed Velocity Estimation on a TMS320C240.* Spra363. Texas Instruments. July 1997.

- *Sensorless Control with Kalman Filter on TMS320 Fixed-Point DSP.* Bpra057. Texas Instruments. July 1997.

- S. Alepuz, V. Delos, J. Mª Horrillo, J. Horrillo y J. Triadó, *Control vectorial de motores de inducción. Aplicación con DSP*. Perspectiva, Mundo Electrónico, Nº 310, Junio 2000.

- S. Arnalte, J. Ramos y J. Sanz Feito. *Control de accionamientos de CA. Técnicas digitales de modulación PWM*. Perspectiva, Mundo Electrónico, Nº 295, Febrero 1999.

- *ACI1-1 System Documentation Variable Speed Control of Single-Phase AC Induction Motor*. Digital Control Systems (DCS) Group, SPRU442, Texas Instruments, August 2001.

- S. Bejerke. *Digital signal processing solutions for motor control using the TMS320F240 DSP-Controller*. SPRA345, Texas Instruments, September 1996.

- *Getting Started with VisSim/Motion*. Visual Solutions.

ANEXO 1: EJEMPLOS DESARROLLADOS

DESCRIPCIÓN:

A continuación se hace un listado de los distintos proyectos que se han desarrollados a lo largo del capítulo 3 para su comprensión. Cuatro de ellos se corresponden a la misma aplicación: Parpadeo del led DS2 de la tarjeta F243 DSK, los cuales se han desarrollado de distinta manera como ejemplo de las distintas formas de atacar un proyecto. En particular dos de ellos se han ejecutado en lenguaje C: uno simplificado con lo estrictamente necesario para la aplicación y otro más completo que sirve como base para el desarrollo de una aplicación más compleja. El tercero se ha desarrollado en lenguaje ensamblador con las opciones mínimamente necesarias para su correcto funcionamiento, y el cuarto es éste último modificado para convertirlo en una aplicación que pueda ser monitorizada en tiempo real. Por último, se incluye un ejemplo sencillo en lenguaje C que se ha empleado en el capítulo 3 para desarrollar ejemplos de las distintas herramientas de depuración, cuya aplicación consiste en hacer los cuadrados sucesivos de un número variable.

Para poner en práctica estos ejemplos sin producirse errores es recomendable no emplear el archivo ".mak" que incluyen, debido a que contienen la información de la ruta de acceso que tenían cada archivo en el PC donde se desarrollaron los ejemplos. Por ello, para emplear estos proyectos en Code Composer, es conveniente generar un nuevo proyecto añadiendo cada uno de los archivos incluidos en cada ejemplo. Recordar que todos los archivos deben encontrarse en la misma carpeta, y que esta carpeta no debe tener un nombre en el que aparezcan espacios ni contenga demasiados caracteres.

- PROYECTO LUZ.MAK EN LENGUAJE C SIMPLIFICADO:

Este proyecto consiste en una aplicación para hacer parpadear el led DS2 de la tarjeta DSK. Se ha desarrollado en C a partir de los siguientes archivos:

- **Luz.mak:** Archivo maestro del proyecto en el que se recoge la configuración y archivos que forman parte del mismo.

- **Luz.cmd:** Archivo de comandos del enlazador para definir el mapa de memoria y sus distintas secciones.

- **Luz.c:** Archivo con la aplicación principal del proyecto.

- **rts2xx.lib:** Librería de entorno en lenguaje C.

Tiene la particularidad de que al no incluirse la tabla de vectores de interrupción, es necesario ejecutar la opción Debug->Restart para llevar el programa a su punto de comienzo en la función "c_int0" después de haber hecho un reset inicial del DSP.

Luz.mak

```
/********* Code Composer V1 Project Data *************/
The following section contains data generated by Code Composer to store project
information like build options, source filenames and dependencies.

[command filename]
luz.cmd 2

[Project Root]
C:\tic2xx\myprojects\Buenos\Luz_corto

[build options]
3
Linker = "-c -o luz.out -x "
Assembler = "-s -v2xx "
Compiler = "-g -v2xx -as –frC:\tic2xx\myprojects\Buenos\Luz_corto "

[source files]
luz.c 1051638854 1
rts2xx.lib 0 1
10201

[dependencies]
 0 -802
 0 -802

[version]
2.0
*/
-c -o luz.out -x
"luz.obj"
"rts2xx.lib"
"luz.cmd"
/** End of Project Data Generated by Code Composer ***/
```

Luz.cmd

```
/*   Spectrum Digital Test code
 *   Copyright (c) 1997.
 *   Spectrum Digital, Inc
 *   ALL RIGHTS RESERVED
*/
/* The memory mapping in this file should match the
 *mapping in the EMUINIT.CMD file in the load directory
*/
 /* linker commands */
-c                  /* ROM autoinitialization */
-x                  /* force rereading libraries */
-o luz.out                      /* output file */
```

```
-m luz.map                    /* map file */

luz.obj
-l rts2xx.lib
                    /* specify memory map */
MEMORY
{
 PAGE 0 :                                                    /* program memory */
                VECS: origin = 00000h, length = 0003Fh
                CODE: origin = 02000h, length = 0FDC0h
 PAGE 1 :                                                     /* data memory */
                Regs       : origin = 00000h, length = 0005Fh
                Ext_Ram : origin = 08000h, length = 07FFFh
}
                     /* specify sections */
SECTIONS
{
 vectors   :   > VECS PAGE = 0
 .cinit     :   > CODE PAGE = 0
 .text      :   > CODE PAGE = 0
 .switch   :   > CODE PAGE = 0
 .const    :   > CODE PAGE = 0
 .data      :   > Ext_Ram PAGE = 1
 .bss       :   > Ext_Ram PAGE = 1
 .heap     :   > Ext_Ram PAGE = 1
 .stack    :   > Ext_Ram PAGE = 1
}
```

Luz.c

```
void main(void)
{
asm("      LDP    #0E0h");              /* Load Data Page(WD)*/
asm("      SPLK   #00EFh,07029h");      /* Desactiva WD */

asm("            CLRC  SXM");           /*Desactiva bit de signo */
asm("START:  SETC   XF");               /* Enciende led (XF=1) */
asm("            LACC   #0FFFEh");      /* Acumulador=FFFFh */

asm("LOOP1:  SUB    #1");              /* Resta 1 */
asm("            BCND   LOOP1, NEQ");   /*Comprueba ACC=0*/
asm("            CLRC   XF");           /* Apaga led (XF=0) */
asm("            LACC   #0FFFFh");      /* Acumulador=FFFFh */

asm("LOOP2:  SUB    #1");              /* Resta 1 */
asm("            BCND   LOOP2, NEQ");   /*Comprueba ACC=0*/
asm("            B      START");        /* Salta a Start */
}
```

- PROYECTO LUZ.MAK EN LENGUAJE C:

El siguiente proyecto es una ampliación del anterior para que pueda ser utilizado como base para el comienzo de una aplicación más compleja en la que se empleen alguno de los periféricos del DSP. Por ello los archivos necesarios para su desarrollo son:

- **Luz.mak:** Archivo maestro del proyecto en el que se recoge la configuración y archivos que forman parte del mismo.

- **Luz.cmd:** Archivo de comandos del enlazador para definir el mapa de memoria y sus distintas secciones.

- **Vectors.asm:** Tabla de vectores de interrupción del DSP.

- **Boot.asm:** Módulo de iniciación del entorno para el DSP.

- **Luz.c:** Archivo con la aplicación principal del proyecto.

- **rts2xx.lib:** Librería de entorno en lenguaje C.

Al incluir la tabla de vectores de interrupción con un reset del DSP se lleva el programa a su punto de comienzo "c_int0".

Luz.mak

/********** Code Composer V1 Project Data ************/
The following section contains data generated by Code Composer to store project information like build options, source filenames and dependencies.

[command filename]
luz.cmd 4

[Project Root]
C:\tic2xx\myprojects\Buenos\Luz

[build options]
3
Linker = "-c -o luz.out -x "
Assembler = "-s -v2xx "
Compiler = "-g -v2xx -as –frC:\tic2xx\myprojects\Buenos\Luz "
[source files]
luz.c 0 1
BOOT.ASM 0 1
vectors.asm 0 1
rts2xx.lib 0 1
-21901

[dependencies]
0 -802
0 -802

```
 0 -802
 0 –802
[version]
2.0
*/
-c -o luz.out -x
"vectors.obj"
"BOOT.obj"
"luz.obj"
"rts2xx.lib"
"luz.cmd"
/** End of Project Data Generated by Code Composer ***/
```

Luz.cmd

```
/*********************************************************/
/*LINKER COMMAND FIL-MEMORY SPECIFICATION  C240/243  */
/*********************************************************/
/***********************************************/
/******** Specify the memory configuration ***********/
/***********************************************/
MEMORY
{
   PAGE 0: VECS:  origin = 00000h, length = 00040h
         FLASH: origin = 00040h, length = 03FC0h
         SARAM: origin = 04000h, length = 00800h
         B0:    origin = 0FF00h, length = 00100h

   PAGE 1: B0:    origin = 00200h, length = 00100h
         B1:    origin = 00300h, length = 00100h
         B2:    origin = 00060h, length = 00020h
         SARAM: origin = 08000h, length = 00800h
}
/*-------------------------------------------------------------*/
/*                SECTIONS    ALLOCATION             */
/*-------------------------------------------------------------*/

SECTIONS
{
    vectors   : { } > VECS    PAGE 0
   .text      : { } > FLASH   PAGE 0
   .cinit     : { } > FLASH   PAGE 0
   .switch    : { } > FLASH   PAGE 0

   .const     : { } > SARAM  PAGE 1
   .data      : { } > SARAM  PAGE 1
   .bss       : { } > SARAM  PAGE 1
   .stack     : { } > SARAM  PAGE 1
   .sysmem : { } > SARAM  PAGE 1
}
```

Vectors.asm

```
*******************************************************
*       Filename:       vectors.asm                   *
*Author:David M. Alter, Texas Instruments Inc.         *
*Last   Modified:       03/14/01                      *
*Description:Interrupt vector table for '240x DSP core *
* for use with assembly language programs.             *
*******************************************************

        .ref _c_int0

        .sect   "vectors"
rset:   B       _c_int0         ;00h reset
int1:   B       int1            ;02h INT1
int2:   B       int2            ;04h INT2
int3:   B       int3                    ;06h INT3
int4:   B       int4            ;08h INT4
int5:   B       int5            ;0Ah INT5
int6:   B       int6            ;0Ch INT6
int7:   B       int7            ;0Eh reserved
int8:   B       int8            ;10h INT8  (software)
int9:   B       int9            ;12h INT9  (software)
int10:  B       int10           ;14h INT10 (software)
int11:  B       int11           ;16h INT11 (software)
int12:  B       int12           ;18h INT12 (software)
int13:  B       int13           ;1Ah INT13 (software)
int14:  B       int14           ;1Ch INT14 (software)
int15:  B       int15           ;1Eh INT15 (software)
int16:  B       int16           ;20h INT16 (software)
int17:  B       int17           ;22h TRAP
int18:  B       int18           ;24h NMI
int19:  B       int19           ;26h reserved
int20:  B       int20           ;28h INT20 (software)
int21:  B       int21           ;2Ah INT21 (software)
int22:  B       int22           ;2Ch INT22 (software)
int23:  B       int23           ;2Eh INT23 (software)
int24:  B       int24           ;30h INT24 (software)
int25:  B       int25           ;32h INT25 (software)
int26:  B       int26           ;34h INT26 (software)
int27:  B       int27           ;36h INT27 (software)
int28:  B       int28           ;38h INT28 (software)
int29:  B       int29           ;3Ah INT29 (software)
int30:  B       int30           ;3Ch INT30 (software)
int31:  B       int31           ;3Eh INT31 (software)
```

Boot.asm

```
;  EVM320C243 Test Code
;  Copyright (c) 1997.
;  Spectrum Digital, Inc.
;  ALL RIGHTS RESERVED
;  This is a modification of the boot routine V6.60
;  from the TI run time support library. This was
;  chosen as a starting point because TI split the C5x
;  and 2x/2xx combination.
;
;  This module contains the following definitions :
;
;       __stack    - Stack memory area
;       _c_int0    - Boot function
;                  _var_init  - Function which processes
;                        initialization tables
;
        .global  _c_int0, cinit
        .global  _main, _abort
        .global .bss, end

WD_CNTR   .set     07023h          ;WD Counter reg
WD_KEY    .set     07025h          ;WD Key reg
WD_CNTL   .set     07029h          ;WD Control reg

;------------------------------------------------------
; Debug directives
;------------------------------------------------------
;                 .def    GPR0            ;General purpose registers.
;                 .def    GPR1
;                 .def    GPR2
;                 .def    GPR3

;------------------------------------------------------
; Variable Declarations for on chip RAM Blocks
;------------------------------------------------------
;                 .bss    GPR0,1          ;General purpose registers.
;                 .bss    GPR1,1
;                 .bss    GPR2,1
;                 .bss    GPR3,1
;                 .bss    REG5,1
;                 .bss    REGA,1

;------------------------------------------------------
; M A C R O - Definitions
;------------------------------------------------------
SBIT0           .macro DMA, MASK ;Clear bit Macro
                LACC DMA
                AND   #(0FFFFh-MASK)
```

```
                SACL  DMA
                .endm

SBIT1           .macro DMA, MASK ;Set bit Macro
                LACC DMA
                OR     #MASK
                SACL  DMA
                .endm

KICK_DOG   .macro                ;Watchdog reset macro
                LDP    #00E0h
                SPLK  #05555h, WD_KEY
                SPLK  #0AAAAh, WD_KEY
                LDP    #0h
                .endm

;
; CONST COPY OPTION
; If your system cannot support allocating an
; initialized section to data memory, and you want the
; boot routine to copy .const from program to data
; memory, then set this CONST_COPY variable to 1
;
; Note the code that does the copy depends on you
; having the following in your linker command file
;
;      MEMORY
;      {
;       PAGE 0 : PROG : ...   /* 'PROG' AND 'DATA'*/
;                                                    /* ARE EXAMPLE NAMES */
;       PAGE 1 : DATA : ...
;       ...
;      }
;
;      SECTIONS
;      {
;       ...
;       .const : load = PROG PAGE 0, run = DATA PAGE 1
;                {
;                 __const_run = . ;
;                 *(.c_mark)
;                 *(.const)
;                 __const_length = . - __const_run;
;                }
;       ...
;      }
;
CONST_COPY         .set      0
;
; For CONST COPY, Define the load address of the .const
```

```
; section depends on linker command file being written
; as above
;
        .if    CONST_COPY
        .sect   ".c_mark"
        .label  __const_load

        .global __const_run, __const_length

        .text
        .endif ; CONST_COPY

;
; Declare the stack.  Size is determined by the linker
; option -stack
;
__stack:       .usect  ".stack",0

;
; FUNCTION DEF : _c_int0
;
;   1) Set up stack
;   2) Set up proper status
;   3) If "cinit" is not -1, init global variables
;   4) call users' program
;
;
_c_int0:              ; entry point from reset vector

     SETC    INTM              ;Disable interrupts
;
;Initialize status bit fields NOT initialized at reset
;
     CLRC    XF                ; turn off xf bit
     CLRC    SXM              ;Clear Sign Extension Mode
     CLRC    OVM              ;Reset Overflow Mode
     CLRC    CNF             ;Config Block B0 to Data mem.

     LDP     #0E0h
     SPLK    #006Fh,WD_CNTL
     KICK_DOG

;    LDP     #00E0h
;         SPLK #00CBh,PLL_CNTL2
                                        ;CLKIN(XTAL)=10MHz,CPUCLK=20MHz
;         SPLK          #00C3h,PLL_CNTL1
                                  ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2,
;    SPLK    #40C0h,SYSCR              ;CLKOUT=CPUCLK,no reset
;    LDP     #0000h
;    SPLK    #4h,GPR3
```

```
;     OUT     GPR3,WSGR ;Set XMIF to run w/no(0) wait
                                                                    states
                ;1 wait state for offchip I/O reads

;
;  Set up initial stack and frame pointers
;
        LRLK    AR0,__stack         ; set up frame pointer
        LRLK    AR1,__stack         ; set up stack pointer
;
; Initialize status bit fields which are set to these
; same values by reset.If you run this routine from
; reset, you can comment out this code.
;
        SPM     0           ; product shift count of zero
    MAR    *,AR0              ; AR = 0, mls 10/07/96

        SSXM              ; set SXM=1 for next instruction
;
; If cinit is not -1, process initialization tables
;
        LALK    cinit         ; get pointer to init tables
        ADDK    1
        BZ      skip              ; if (cinit == -1)

        CALL    _var_init,AR1        ; var_init()
;
;  Call the user's program
;
skip:
        .if     CONST_COPY
        CALL    const_copy
        .endif

                    ; start of hw stack init code
      LACK    #0014h     ; vectors for underflow
      SACL    *
      PSHD    *
      PSHD    *
      PSHD    *
      PSHD    *
      PSHD    *
      PSHD    *
      PSHD    *
      PSHD    *

        CALL    _main,AR1
        CALL    _abort,AR1           ; to never return...

        .page
```

```
;
;  FUNCTION DEF : _var_init
;
;  PROCESS INITIALIZATION TABLES.  TABLES ARE IN
;  PROGRAM MEMORY IN THE FOLLOWING FORMAT :
;
;       .word  <length of init data in words>
;       .word  <address of variable to initialize>
;       .word  <init data>
;       .word  ...
;
;  The init table is terminated with a zero length
;

_var_init:

; C2xx Version
;
        ADRK    2      ; allocate two words of local memory
        LALK    cinit ; load accumulator with base of table
        LARP    AR0
;
;  Read init record header.
;An init record with a zero length terminates the list.
;
loop:
        TBLR    *+                  ; read length
        ADDK    1
        TBLR    *                   ; read address

        LAR     AR2,*-     ; load variable address into ar2
        LAR     AR3,*,AR3       ; load count into ar3
        BANZ    copy,*-,AR2        ; check for end of table
;
;  At end of list, return to caller
;
        LARP    AR1
        SBRK    2                   ; deallocate locals
        RET                        ; return to _c_int0
;
;  Perform the copy of data from program to data
;
copy:
        ADDK    1              ; increment pointer to data
        TBLR    *+,AR3 ; copy data from program to variable
        BANZ    copy,*-,AR2           ; until count is zero

        ADDK    1       ; point to beginning of next record
        B       loop,AR0         ; go process next record
```

```
        .page

; CONST COPY Routine - copies the .const section from
; program to data memory
;
        .if CONST_COPY
const_copy:

;
; C2xx version - must use 'RPT *' because RPTK count
; isn't big enough
;
        LALK    #__const_length ;load length of const
                                                        ; section
        BZ      quit                ; if 0, quit
        LRLK    AR2,#__const_run        ; AR2 = const
                                                    ;address in data
        LALK    #__const_length-1       ; load length - 1
        SACL    *                   ; write to temp

        RPT     *,AR2               ; repeat length times
        BLKP    #__const_load,*+  ; block copy from program

        LARP    AR1                 ; restore ARP to SP
quit:
        RET                         ; return

        .endif ; CONST_COPY

        .end
```

Luz.c

```
void main(void)
{
asm("             CLRC  SXM");              /*Desactiva bit de signo */
asm("START:  SETC    XF");                  /* Enciende led (XF=1) */
asm("             LACC    #0FFFEh");        /* Acumulador=FFFFh */

asm("LOOP1:  SUB     #1");                  /* Resta 1 */
asm("             BCND    LOOP1, NEQ");    /*Comprueba ACC=0*/
asm("             CLRC    XF");            /* Apaga led (XF=0) */
asm("             LACC    #0FFFFh");       /* Acumulador=FFFFh */

asm("LOOP2:  SUB     #1");                  /* Resta 1 */
asm("             BCND    LOOP2, NEQ");    /*Comprueba ACC=0*/
asm("             B       START");         /* Salta a Start */

}
```

- PROYECTO LED.MAK EN LENGUAJE ENSAMBLADOR:

Variación del programa anterior simplificado para su implementación en lenguaje ensamblador. Los archivos necesarios para el proyecto son:

- **Led.mak:** Archivo maestro del proyecto en el que se recoge la configuración y archivos que forman parte del mismo.

- **Luz.cmd:** Archivo de comandos del enlazador para definir el mapa de memoria y sus distintas secciones.

- **Vectors.asm:** Tabla de vectores de interrupción del DSP.

- **Led.asm:** Archivo con la aplicación principal del proyecto.

Led.mak

```
/*********** Code Composer V1 Project Data ************
The following section contains data generated by Code Composer to store project
information like build options, source filenames and dependencies.

[command filename]
luz.cmd 2

[Project Root]
C:\tic2xx\myprojects\Buenos\Led_asm

[build options]
3
Linker = "-c -o led.out -x "
Assembler = "-s -v2xx "
Compiler = "-g -v2xx -as -frC:\tic2xx\myprojects\Buenos\Led_asm "

[source files]
led.asm 1051636708 1
vectors.asm 1051632050 1
138

[dependencies]
 0 -802
 0 -802

[version]
2.0
*/
-c -o led.out -x
"vectors.obj"
"led.obj"
"luz.cmd"
/** End of Project Data Generated by Code Composer ***/
```

Luz.cmd

```
/*************************************************************/
/*  LINKER COMMAND FILE-MEMORY SPECIFICATION C240/243 */
/*************************************************************/
/*********************************************************/
/******** Specify the memory configuration ***********/
/*********************************************************/
MEMORY
{
   PAGE 0:              VECS:     origin = 00000h, length = 00040h
                        FLASH:  origin = 00040h, length = 03FC0h
                        SARAM: origin = 04000h, length = 00800h
                        B0:         origin = 0FF00h, length = 00100h

   PAGE 1:               B0:    origin = 00200h, length = 00100h
                         B1:    origin = 00300h, length = 00100h
                         B2:    origin = 00060h, length = 00020h
                        SARAM: origin = 08000h, length = 00800h
}
/*-----------------------------------------*/
/*      SECTIONS ALLOCATION      */
/*-----------------------------------------*/

SECTIONS
{
    vectors   : { } > VECS    PAGE 0
   .text        : { } > FLASH   PAGE 0
   .cinit       : { } > FLASH   PAGE 0
   .switch     : { } > FLASH   PAGE 0

   .const       : { } > SARAM   PAGE 1
   .data        : { } > SARAM   PAGE 1
   .bss         : { } > SARAM   PAGE 1
   .stack       : { } > SARAM   PAGE 1
   .sysmem   : { } > SARAM   PAGE 1
}
```

Vectors.asm

```
       .ref _c_int0

      .sect   "vectors"
rset:  B      _c_int0      ;00h reset
int1:  B      int1         ;02h INT1
int2:  B      int2         ;04h INT2
int3:  B      int3         ;06h INT3
int4:  B      int4         ;08h INT4
int5:  B      int5         ;0Ah INT5
int6:  B      int6         ;0Ch INT6
int7:  B      int7         ;0Eh reserved
```

```
int8:   B      int8          ;10h INT8  (software)
int9:   B      int9          ;12h INT9  (software)
int10:  B      int10         ;14h INT10 (software)
int11:  B      int11         ;16h INT11 (software)
int12:  B      int12         ;18h INT12 (software)
int13:  B      int13         ;1Ah INT13 (software)
int14:  B      int14         ;1Ch INT14 (software)
int15:  B      int15         ;1Eh INT15 (software)
int16:  B      int16         ;20h INT16 (software)
int17:  B      int17         ;22h TRAP
int18:  B      int18         ;24h NMI
int19:  B      int19         ;26h reserved
int20:  B      int20         ;28h INT20 (software)
int21:  B      int21         ;2Ah INT21 (software)
int22:  B      int22         ;2Ch INT22 (software)
int23:  B      int23         ;2Eh INT23 (software)
int24:  B      int24         ;30h INT24 (software)
int25:  B      int25         ;32h INT25 (software)
int26:  B      int26         ;34h INT26 (software)
int27:  B      int27         ;36h INT27 (software)
int28:  B      int28         ;38h INT28 (software)
int29:  B      int29         ;3Ah INT29 (software)
int30:  B      int30         ;3Ch INT30 (software)
int31:  B      int31         ;3Eh INT31 (software)
```

Led.asm

```
        .global  _c_int0              ; Global declaration
WD_CNTL   .set  07029h               ;WD Control register
        .text                        ; Code in text section

_c_int0:            ; entry point from reset vector

          CLRC    SXM          ;Clear Sign Extension Mode

          LDP     #0E0h                ; Load Data Page (WD)
          SPLK    #00EFh,WD_CNTL   ; Disable watchdog

start:                                      ; Aplicación
          SETC XF                    ; Enciende led (XF=1)
          LACC #0FFFFh             ; Acumulador= FFFFh

loop1:    SUB #1                   ; Resta 1 al Acumulador
          BCND loop1, NEQ          ; Comprueba si resultado=0
          CLRC XF                       ; Apaga led (XF=0)
          LACC #0FFFFh             ; Acumulador= FFFFh

loop2:    SUB #1                    ; Resta 1 al Acumulador
          BCND loop2, NEQ     ; Comprueba si resultado=0
          B start                  ; Salta a etiqueta Start
```

- PROYECTO LED.MAK EN TIEMPO REAL:

Modificación del proyecto anterior para su depuración en tiempo real, por lo que consta de los siguientes archivos:

- **Led.mak:** Archivo maestro del proyecto en el que se recoge la configuración y archivos que forman parte del mismo.

- **Luz.cmd:** Archivo modificado de comandos del enlazador para definir el mapa de memoria y sus distintas secciones en tiempo real.

- **Rtectors.asm:** Tabla de vectores de interrupción del DSP modificada para tiempo real.

- **C200mnrt.asm:** Programa monitor en tiempo real.

- **C200mnrt.i:** Archivo de configuración del programa monitor.

- **Led.asm:** Archivo con la aplicación principal del proyecto modificada para tiempo real.

Los archivos c200mnrt.asm y c200mnrt.i no se listan a continuación debido a su extensión. Éstos se encuentran en el directorio MONITOR de Code Composer.

Led.mak

```
/********* Code Composer V1 Project Data **************
  The following section contains data generated by Code Composer to store project
information like build options, source filenames and dependencies.

[command filename]
luz.cmd 4

[Project Root]
C:\tic2xx\myprojects\Buenos\Luz_RTM

[build options]
3
Linker = "-c -o led.out -x "
Assembler = "-s -v2xx "
Compiler = "-g -v2xx -as -frC:\tic2xx\myprojects\Buenos\Luz_RTM "

[source files]
led.asm 1051655756 1
RTVECTOR.ASM 1051639934 1
C200MNRT.ASM 943958348 1
c200mnrt.i 943958350 0
-8584
```

```
[dependencies]
 0 -802
 1 3:10 -16814
 1 3:30 19619
 0 -802

[version]
2.0
*/
-c -o led.out -x
"C200MNRT.obj"
"RTVECTOR.obj"
"led.obj"
"luz.cmd"
/** End of Project Data Generated by Code Composer ***/
```

Luz.cmd

```
/*********************************************************/
/*  LINKER COMMAND FILE-MEMORY SPECIFICATIONC240/243 */
/*************** Specify the memory configuration ***************/
/*********************************************************/
MEMORY
{
   PAGE 0:              VECS:   origin = 00000h, length = 00040h
                        FLASH:  origin = 00040h, length = 03FC0h
                        SARAM: origin = 04000h, length = 00800h
                         B0:        origin = 0FF00h, length = 00100h

   PAGE 1:               B0:    origin = 00200h, length = 00100h
                         B1:    origin = 00300h, length = 00100h
                         B2:    origin = 00060h, length = 00020h
                        SARAM: origin = 08000h, length = 00800h
}
/*-------------------------------------------------*/
/* SECTIONS ALLOCATION                             */
/*-------------------------------------------------*/
SECTIONS
{
    vectors       : { } > VECS    PAGE 0
   .text          : { } > FLASH   PAGE 0
   mon_main       : { } > FLASH   PAGE 0

   .data          : { } > SARAM   PAGE 1
   .bss           : { } > B0                PAGE 1
   .stack         : { } > SARAM   PAGE 1
   mom_pge0       : { } > B2      PAGE 1
   mom_rgst       : { } > B2      PAGE 1
   blk_bo         : { } > B0      PAGE 1
}
```

Rtvectors.Asm

```
            .include        "c200mnrt.i" ; Include conditional
                                         ;assembly options.
            .mmregs          ; Include standard register
                             ;mnemonics.

            .global _c_int0, PHANTOM

        .sect "vectors"

RESET   B       _c_int0         ; 00
INT1    B       PHANTOM   ; 02
INT2    B       PHANTOM   ; 04
INT3    B       PHANTOM   ; 06
INT4    B       PHANTOM   ; 08
INT5    B       PHANTOM   ; 0A
INT6    B       PHANTOM   ; 0C
        .if ( 1 ) ; macro occupies fourteen words
                                         ; in the vector table.
MON_EINTR   mon_eintr_vecs      ; 0E
                        ; 10
                        ; 12
                        ; 14
                        ; 16
                        ; 18
                        ; 1A
        .else ; macro not in vector table.
MON_EINTR_B B    MON_EINTR      ; 0E
HUNG10    B    HUNG10         ; 10
HUNG12    B    HUNG12         ; 12
HUNG14    B    HUNG14         ; 14
HUNG16    B    HUNG16         ; 16
HUNG18    B    HUNG18         ; 18
HUNG1A    B    HUNG1A          ; 1A
        .endif
HUNG1C    B     HUNG1C         ; 1C
HUNG1E    B     HUNG1E         ; 1E
HUNG20    B    HUNG20         ; 20
TRAP          B    TRAP          ; 22
NMI           B    PHANTOM          ; 24
        .if ( 1 ) ; macro occupies eight words in
                                         ;the vector table.
MON_ETRAP   mon_etrap_vecs      ; 26
                        ; 28
                        ; 2A
                        ; 2C
        .else ; macro not in vector table.
MON_ETRAP_B B    MON_ETRAP      ; 26
```

```
HUNG28      B    HUNG28        ; 28
HUNG2A      B    HUNG2A        ; 2A
HUNG2C      B    HUNG2C        ; 2C
        .endif
HUNG2E      B    HUNG2E        ; 2E
HUNG30      B    HUNG30        ; 30
HUNG32      B    HUNG32        ; 32
HUNG34      B    HUNG34        ; 34
HUNG36      B    HUNG36        ; 36
HUNG38      B    HUNG38        ; 38
HUNG3A      B    HUNG3A        ; 3A
HUNG3C      B    HUNG3C        ; 3C
HUNG3E      B    HUNG3E        ; 3E
        .end
```

Led.asm

* Step 1: Global Declaration:

```
        .global  _c_int0, PHANTOM
        .global MON_RT_CNFG
```

* Variables declaration:

```
WD_CNTL    .set 07029h          ;WD Control register
IMR        .set 0004h           ;Interrupt Mask Register
IFR        .set 0006h           ;Interrupt Flag Register

   .bss test,1          ;Declara señal test en bss
```

* Main Program:

```
   .text          ;Código en sección text

_c_int0:          ;Entry point from reset vector

        CLRC    SXM        ;Clear Sign Extension Mode

        LDP     #0E0h      ; Load Data Page (WD)
        SPLK  #006Fh, WD_CNTL  ; Disable WatchDog

        LDP     #04h
        SPLK #01h,test    ; Inicializa variable test=1
```

* Step 3: Initialize real-time monitor

```
     CALL MON_RT_CNFG; Call real time monitor routine
```

* Step 4: Unmask interrupt

```
      LDP       #0h
      SPLK      #01000000b,IMR   ;Unmask interrupt 7
                                 ;for real time interrupt
```

* Aplicación:

```
start:
            SETC XF          ;Enciende led
            LDP    #04h
            SPLK #01h,test   ;test=1

            LACC #0FFFFh             ;Acumulador=FFFFh
loop1:  SUB #1                  ;Resta 1
            BCND loop1, NEQ    ;Ve si resultado=0
            CLRC XF            ;Apaga led

            LDP    #04h
            SPLK #0h,test       ;test=0

            LACC #0FFFFh             ;Acumulador=FFFFh
loop2:  SUB #1                  ;Resta 1
            BCND loop2, NEQ    ;Ve si resultado=0

            B start          ;Vuelve al comienzo
```

* Rutina de interrupción PHANTOM:

```
PHANTOM:
            RET              ;Vuelve al programa
```

- PROYECTO CUADRADO.MAK EN LENGUAJE C:

Proyecto en el que una variable i toma valores de 0 a 100 y cuyo cuadrado se almacena en otra k.

- **Cuadrado.mak:** Archivo maestro del proyecto en el que se recoge la configuración y archivos que forman parte del mismo.

- **Cuadrado.cmd:** Archivo de comandos del enlazador para definir el mapa de memoria y sus distintas secciones.

- **Vectors.asm:** Tabla de vectores de interrupción del DSP.

- **Boot.asm:** Módulo de iniciación del entorno para el DSP.

- **Cuadrado.c:** Archivo con la aplicación principal del proyecto.

- **rts2xx.lib:** Librería de entorno en lenguaje C.

Los archivos Boot.asm y cuadrado.cmd no se listan a continuación al ser exactamente los mismos que los empleados en los anteriores proyectos en lenguaje C.

Cuadrado.mak

```
/************ Code Composer V1 Project Data **********
The following section contains data generated by Code Composer to store project
information like build options, source filenames and dependencies.

[command filename]
cuadrado.cmd 4

[Project Root]
C:\tic2xx\myprojects\Buenos\Cuadrado

[build options]
3
Linker = "-c -o cuadrado.out -x "
Assembler = "-s -v2xx "
Compiler = "-g -v2xx -as –frC:\tic2xx\myprojects\Buenos\Cuadrado "

[source files]
cuadrado.c 0 1
BOOT.ASM 0 1
vectors.asm 0 1
rts2xx.lib 0 1
-9791

[dependencies]
```

```
 0 -802
 0 -802
 0 -802
 0 -802
[version]
2.0
*/
-c -o cuadrado.out -x
"vectors.obj"
"BOOT.obj"
"cuadrado.obj"
"rts2xx.lib"
"cuadrado.cmd"
/** End of Project Data Generated by Code Composer ***/
```

Vectors.asm

```
        .title          "vectors.asm"
        .ref            _c_int0,_c_dummy1
        .sect           ".vectors"

reset:      b           _c_int0
int1:       b           _c_dummy1
int2:       b           _c_dummy1
int3:       b           _c_dummy1
int4:       b           _c_dummy1
int5:       b           _c_dummy1
int6:       b           _c_dummy1
reserved:   b           _c_dummy1
sw_int8:    b           _c_dummy1
sw_int9:    b           _c_dummy1
sw_int10:   b           _c_dummy1
sw_int11:   b           _c_dummy1
sw_int12:   b           _c_dummy1
sw_int13:   b           _c_dummy1
sw_int14:   b           _c_dummy1
sw_int15:   b           _c_dummy1
sw_int16:   b           _c_dummy1
trap:       b           _c_dummy1
nmint:      b           _c_dummy1
emu_trap:   b           _c_dummy1
sw_int20:   b           _c_dummy1
sw_int21:   b           _c_dummy1
sw_int22:   b           _c_dummy1
sw_int23:   b           _c_dummy1
```

Cuadrado.c

```
void c_dummy1 (void)
{                                   /* función para atrapar funciones espúreas */
  while (1);
}

void main (void)                    /* Función principal */
{
    unsigned int i,k,m=100;     /* Iniciación de variables */
    while(1)
     {
      for(i=0;i<m;i++)        /* Bucle para el cálculo del cuadrado de "i" */
       k=i*i;
     }
}
```

ANEXO 2: REGISTROS PROGRAMABLES EN EL TMS320F243

A continuación se detallan todos los periféricos del procesador TMS320F243 que son susceptibles de ser programados en una aplicación, indicando su nombre y dirección en el mapa de memoria del procesador. Posteriormente se detalla el archivo "regs243.h" en el que se definen las etiquetas para estos registros para su empleo en los ejemplos desarrollados a lo largo del capítulo 4.También se da el listado del archivo "mio.gel" de inicio para Code Composer comentado en el capítulo 3, el cual genera menús de acceso rápido a los registros de los periféricos del procesador en el menú "GEL" de la barra superior, de forma que sea más sencillo y rápido su inclusión en la ventana "Watch Window" para su monitorización.

Dirección	Registro	Descripción
Interna	ST0	CPU Status Register 0
Interna	ST1	CPU Status Register 1
0004h	IMR	CPU Interrupt Mask Register
0005h	GREG	Global Data Memory Configuration Register
0006h	IFR	CPU Interrupt Flag Register
7010h	PIRQR0	Peripheral Interrupt Request Register 0
7011h	PIRQR1	Peripheral Interrupt Request Register 1
7014h	PIACKR0	Peripheral Interrupt Acknowledge Register 0
7015h	PIACKR1	Peripheral Interrupt Acknowledge Register 1
7018h	SCSR	System Control and Status Register
701Ch	DINR	Device Identification Name Register
701Eh	PIVR	Peripheral Interrupt Vector Register
7023h	WDCNTR	Watchdog Counter Register
7025h	WDKEY	Watchdog Reset Key Register
7029h	WDCR	Watchdog Timer Control Register
7032h	ADCTRL1	ADC Control Register 1
7034h	ADCTRL2	ADC Control Register 2
7036h	ADCFIFO1	ADC Data Register FIFO 1
7038h	ADCFIFO2	ADC Data Register FIFO 2

Dirección	Registro	Descripción
7040h	SPICCR	SPI Configuration Control Register
7041h	SPICTL	SPI Operation Control Register
7042h	SPISTS	SPI Status Register
7044h	SPIBRR	SPI Baud Rate Control Register
7046h	SPIRXEMU	SPI Emulation Buffer Register
7047h	SPIRXBUF	SPI Serial Receive Buffer Register
7048h	SPITXBUF	SPI Serial Transmit Buffer Register
7049h	SPIDAT	SPI Serial Data Register
704Fh	SPIPRI	SPI Priority Control Register
7050h	SCICCR	SCI Communication Control Register
7051h	SCICTL1	SCI Control Register 1
7052h	SCIHBAUD	SCI Baud-Select Register, High Bits
7053h	SCILBAUD	SCI Baud-Select Register, Low Bits
7054h	SCICTL2	SCI Control Register 2
7055h	SCIRXST	SCI Receiver Status Register
7056h	SCIRXEMU	SCI Emulation Data Buffer Register
7057h	SCIRXBUF	SCI Receiver Data Buffer Register
7059h	SCITXBUF	SCI Transmit Data Buffer Register
705Fh	SCIPRI	SCI Priority Control Register
7070h	XINT1CR	External Interrupt 1 Control Register
7071h	XINT2CR	External Interrupt 2 Control Register
7090h	OCRA	I/O Mux Control Register A
7092h	OCRB	I/O Mux Control Register B
7098h	PADATDIR	I/O Port A Data and Direction Register
709Ah	PBDATDIR	I/O Port B Data and Direction Register

Dirección	Registro	Descripción
709Ch	PCDATDIR	I/O Port C Data and Direction Register
709Eh	PDDATDIR	I/O Port D Data and Direction Register
7100h	MDER	Mailbox Direction/Enable Register
7101h	TCR	Transmission Control Register
7102h	RCR	Receive Control Register
7103h	MCR	Master Control Register
7104h	BCR2	Bit Configuration Register 2
7105h	BCR1	Bit Configuration Register 1
7106h	ESR	Error Status Register
7107h	GSR	Global Status Register
7108h	CEC	CAN Error Counter Registers
7109h	CAN_IFR	Interrupt Flag Register
710Ah	CAN_IMR	Global Interrupt Mask Register
710Bh	LAM0_H	Local Acceptance Mask Mailbox 0 and 1
710Ch	LAM0_L	Local Acceptance Mask Mailbox 0 and 1
710Dh	LAM1_H	Local Acceptance Mask Mailbox 2 and 3
710Eh	LAM1_L	Local Acceptance Mask Mailbox 2 and 3
7200h	MSGID0L	CAN Message ID for Mailbox 0 (lower 16 bits)
7201h	MSGID0H	CAN Message ID for Mailbox 0 (upper 16 bits)
7202h	MSGCTRL0	CAN Message Control Field 0
7204h	MBX0A	CAN 2 of 8 Bytes of Mailbox 0
7205h	MBX0B	CAN 2 of 8 Bytes of Mailbox 0
7206h	MBX0C	CAN 2 of 8 Bytes of Mailbox 0
7207h	MBX0D	CAN 2 of 8 Bytes of Mailbox 0

Dirección	Registro	Descripción
7208h	MSGID1L	CAN Message ID for Mailbox 1 (lower 16 bits)
7209h	MSGID1H	CAN Message ID for Mailbox 1 (upper 16 bits)
720Ah	MSGCTRL1	CAN Message Control Field 1
720Ch	MBX1A	CAN 2 of 8 Bytes of Mailbox 1
720Dh	MBX1B	CAN 2 of 8 Bytes of Mailbox 1
720Eh	MBX1C	CAN 2 of 8 Bytes of Mailbox 1
720Fh	MBX1D	CAN 2 of 8 Bytes of Mailbox 1
7210h	MSGID2L	CAN Message ID for Mailbox 2 (lower 16 bits)
7211h	MSGID2H	CAN Message ID for Mailbox 2 (upper 16 bits)
7212h	MSGCTRL2	CAN Message Control Field 2
7214h	MBX2A	CAN 2 of 8 Bytes of Mailbox 2
7215h	MBX2B	CAN 2 of 8 Bytes of Mailbox 2
7216h	MBX2C	CAN 2 of 8 Bytes of Mailbox 2
7217h	MBX2D	CAN 2 of 8 Bytes of Mailbox 2
7218h	MSGID3L	CAN Message ID for Mailbox 3 (lower 16 bits)
7219h	MSGID3H	CAN Message ID for Mailbox 3 (upper 16 bits)
721Ah	MSGCTRL3	CAN Message Control Field 3
721Ch	MBX3A	CAN 2 of 8 Bytes of Mailbox 3
721Dh	MBX3B	CAN 2 of 8 Bytes of Mailbox 3
721Eh	MBX3C	CAN 2 of 8 Bytes of Mailbox 3
721Fh	MBX3D	CAN 2 of 8 Bytes of Mailbox 3
7220h	MSGID4L	CAN Message ID for Mailbox 4 (lower 16 bits)

Dirección	Registro	Descripción
7221h	MSGID4H	CAN Message ID for Mailbox 4 (upper 16 bits)
7222h	MSGCTRL4	CAN Message Control Field 4
7224h	MBX4A	CAN 2 of 8 Bytes of Mailbox 4
7225h	MBX4B	CAN 2 of 8 Bytes of Mailbox 4
7226h	MBX4C	CAN 2 of 8 Bytes of Mailbox 4
7227h	MBX4D	CAN 2 of 8 Bytes of Mailbox 4
7228h	MSGID5L	CAN Message ID for Mailbox 5 (lower 16 bits)
7229h	MSGID5H	CAN Message ID for Mailbox 5 (upper 16 bits)
722Ah	MSGCTRL5	CAN Message Control Field 5
722Ch	MBX5A	CAN 2 of 8 Bytes of Mailbox 5
722Dh	MBX5B	CAN 2 of 8 Bytes of Mailbox 5
722Eh	MBX5C	CAN 2 of 8 Bytes of Mailbox 5
722Fh	MBX5D	CAN 2 of 8 Bytes of Mailbox 5
7400h	GPTCON	GP Timer Control Register
7401h	T1CNT	GP Timer 1 Counter Register
7402h	T1CMPR	GP Timer 1 Compare Register
7403h	T1PR	GP Timer 1 Period Register
7404h	T1CON	GP Timer 1 Control Register
7405h	T2CNT	GP Timer 2 Counter Register
7406h	T2CMPR	GP Timer 2 Compare Register
7407h	T2PR	GP Timer 2 Period Register
7408h	T2CON	GP Timer 2 Control Register
7411h	COMCON	Compare Control Register
7413h	ACTR	Full-Compare Action Control Register

Dirección	Registro	Descripción
7415h	DBTCON	Dead-Band Timer Control Register
7417h	CMPR1	Full-Compare Unit Compare Register 1
7418h	CMPR2	Full-Compare Unit Compare Register 2
7419h	CMPR3	Full-Compare Unit Compare Register 3
7420h	CAPCON	Capture Control Register
7422h	CAPFIFO	Capture FIFO Status Register
7423h	CAP1FIFO	Two-Level-Deep Capture FIFO Stack 1
7424h	CAP2FIFO	Two-Level-Deep Capture FIFO Stack 2
7425h	CAP3FIFO	Two-Level-Deep Capture FIFO Stack 3
7427h	CAP1FBOT	Capture 1 FIFO Bottom Stack Register
7428h	CAP2FBOT	Capture 2 FIFO Bottom Stack Register
7429h	CAP3FBOT	Capture 3 FIFO Bottom Stack Register
742Ch	EVIMRA	EV Interrupt Mask Register A
742Dh	EVIMRB	EV Interrupt Mask Register B
742Eh	EVIMRC	EV Interrupt Mask Register C
742Fh	EVIFRA	Interrupt Flag Register A
7430h	EVIFRB	Interrupt Flag Register B
7431h	EVIFRC	Interrupt Flag Register C
FF0Fh (I/O space)	FCMR	Flash Control Mode Register
FFFFh	WSGR	Wait State Generator Register

- Contenido del archivo “regs243.h”:

```
#define IMR    *(volatile unsigned int *)0x0004      /* CPU Interrupt Mask Register */
#define GREG *(volatile unsigned int *)0x0005    /* Global Data Memory
                                                        Configuration Register */
#define IFR     *(volatile unsigned int *)0x0006     /* CPU Interrupt Flag Register */

#define PIRQR0 *(volatile unsigned int *)0x7010    /* Peripheral Interrupt Request
                                                              Register 0 */
#define PIRQR1 *(volatile unsigned int *)0x7011    /* Peripheral Interrupt Request
                                                               Register 1 */
#define PIACKR0 *(volatile unsigned int *)0x7014  /* Peripheral Interrupt
                                                        Acknowledge Register 0 */
#define PIACKR1 *(volatile unsigned int *)0x7015 /* Peripheral Interrupt
                                                        Acknowledge Register 1 */
#define SCSR  *(volatile unsigned int *)0x7018      /* System Control and Status
                                                               Register */
#define DINR  *(volatile unsigned int *)0x701C     /* Device Identification Name
                                                               Register */
#define PIVR  *(volatile unsigned int *)0x701E      /* Peripheral Interrupt Vector
                                                               Register */
#define WDCNTR *(volatile unsigned int *)0x7023   /* Watchdog Counter Register */
#define WDKEY *(volatile unsigned int *)0x7025   /* Watchdog Reset Key Register */
#define WDCR *(volatile unsigned int *)0x7029 /* Watchdog Timer Control Register*/
#define ADCTRL1  *(volatile unsigned int *)0x7032      /* ADC Control Register 1 */
#define ADCTRL2  *(volatile unsigned int *)0x7034      /* ADC Control Register 2 */
#define ADCFIFO1 *(volatile unsigned int *)0x7036      /* ADC Data Register FIFO1*/
#define ADCFIFO2    *(volatile unsigned int *)0x7038   /* ADC Data Register FIFO2*/
#define SPICCR        *(volatile unsigned int *)0x7040    /* SPI Configuration Control
                                                               Register */
#define SPICTL  *(volatile unsigned int *)0x7041  /* SPI Operation Control Register */
#define SPISTS   *(volatile unsigned int *)0x7042     /* SPI Status Register */
#define SPIBRR *(volatile unsigned int *)0x7044 /*SPI Baude Rate Control Register */
#define SPIRXEMU  *(volatile unsigned int *)0x7046    /* SPI Emulation Buffer
                                                               Register */
#define SPIRXBUF   *(volatile unsigned int *)0x7047     /* SPI Serial Receive Buffer
                                                               Register */
#define SPITXBUF    *(volatile unsigned int *)0x7048      /* SPI Serial Transmit Buffer
                                                               Register */
#define SPIDAT        *(volatile unsigned int *)0x704   /* SPI Serial Data Register */
#define SPIPRI    *(volatile unsigned int *)0x704F  /* SPI Priority Control Register */
#define SCICCR    *(volatile unsigned int *)0x705  /* SCI Communication Control
                                                               Register */
#define SCICTL1       *(volatile unsigned int *)0x7051    /* SCI Control Register 1 */
#define SCIHBAUD  *(volatile unsigned int *)0x7052      /* SCI Baud-Select Register,
                                                               High-Bits */
#define SCILBAUD   *(volatile unsigned int *)0x7053    /* SCI Baud-Select Register,
                                                               Low-Bits */
#define SCICTL2       *(volatile unsigned int *)0x7054    /* SCI Control Register 2 */
#define SCIRXST *(volatile unsigned int *)0x7055/* SCI Receiver Status Register */
```

```
#define SCIRXEMU   *(volatile unsigned int *)0x7056   /* SCI Emulation Data Buffer
                                                          Register */
#define SCIRXBUF   *(volatile unsigned int *)0x7057    /* SCI Receicer Data Buffer
                                                          Register */
#define SCITXBUF   *(volatile unsigned int *)0x7059    /* SCI Transmit Data Buffer
                                                          Register */
#define SCIPRI    *(volatile unsigned int *)0x705F    /* SCI Priority Control Register */

#define XINT1CR *(volatile unsigned int *)0x7070          /* External Interrupt 1
                                                          Control Register*/
#define XINT2CR *(volatile unsigned int *)0x7071          /* External Interrupt 2
                                                          Control Register*/
#define OCRA     *(volatile unsigned int *)0x7090     /* I/O Mux Control Register A */
#define OCRB      *(volatile unsigned int *)0x7092    /* I/O Mux Control Register B */
#define PADATDIR  *(volatile unsigned int *)0x7098    /* I/O Port A Data and
                                                          Direction Register */
#define PBDATDIR  *(volatile unsigned int *)0x709a    /* I/O Port B Data and
                                                          Direction Register */
#define PCDATDIR  *(volatile unsigned int *)0x709c    /* I/O Port C Data and
                                                          Direction Register */
#define PDDATDIR  *(volatile unsigned int *)0x709e    /* I/O Port D Data and
                                                          Direction Register */
#define CAN_MDER  *(volatile unsigned int *)0x7100   /* CAN Mailbox
                                                        Direction/Enable Register  */
#define CAN_TCR    *(volatile unsigned int *)0x7101    /* CAN Transmission Control
                                                          Register */
#define CAN_RCR    *(volatile unsigned int *)0x7102    /* CAN Receicer Control
                                                          Register */
#define CAN_MCR   *(volatile unsigned int *)0x7103    /* CAN MAster Control
                                                          Register */
#define CAN_BCR2  *(volatile unsigned int *)0x7104    /* CAN Bit Configuration
                                                          Register 2 */
#define CAN_BCR1  *(volatile unsigned int *)0x7105    /* CAN Bit Configuration
                                                          Register 1 */
#define CAN_ESR    *(volatile unsigned int *)0x7106  /* CAN Error Status Register */
#define CAN_GSR *(volatile unsigned int *)0x7107    /* CAN Global Status Register*/
#define CAN_CEC     *(volatile unsigned int *)0x7108    /* CAN Error Counter
                                                          Register */
#define CAN_IFR *(volatile unsigned int *)0x7109    /* CAN Interrupt Flag Register */
#define CAN_IMR *(volatile unsigned int *)0x710a  /*CAN Interrupt Mask Register*/
#define CAN_LAM0_H *(volatile unsigned int *)0x710b    /* CAN Local Acceptance
                                                        Mask MB0 and MB1 high */
#define CAN_LAM0_L *(volatile unsigned int *)0x710c  /* CAN Local Acceptance
                                                         Mask MB0 and MB1 low  */
#define CAN_LAM1_H *(volatile unsigned int *)0x710d  /* CAN Local Acceptance
                                                        Mask MB3 and MB2 high */
#define CAN_LAM1_L  *(volatile unsigned int *)0x710e   /* CAN Local Acceptance
                                                        Mask MB3 and MB2 low  */
```

```
#define CAN_MSGID0L *(volatile unsigned int *)0x7200/* CAN Message ID for
                                                        MB0 , low */
#define CAN_MSGID0H     *(volatile unsigned int *)0x7201     /* CAN Message ID
                                                        for MB0 , high */
#define CAN_MSGCTRL0 *(volatile unsigned int *)0x7202       /* CAN Message
                                                        Control Field 0 */
#define CAN_MBX0A *(volatile unsigned int *)0x7204   /* CAN 2 of 8 Bytes of
                                                        Mailbox 0 */
#define CAN_MBX0B *(volatile unsigned int *)0x7205   /* CAN 2 of 8 Bytes of
                                                        Mailbox 0 */
#define CAN_MBX0C *(volatile unsigned int *)0x7206   /* CAN 2 of 8 Bytes of
                                                        Mailbox 0 */
#define CAN_MBX0D *(volatile unsigned int *)0x7207   /* CAN 2 of 8 Bytes of
                                                        Mailbox 0 */
#define CAN_MSGID1L  *(volatile unsigned int *)0x7208        /* CAN Message ID
                                                        for MB1 , low */
#define CAN_MSGID1H     *(volatile unsigned int *)0x7209     /* CAN Message ID
                                                        for MB1 , high */
#define CAN_MSGCTRL1 *(volatile unsigned int *)0x720A       /* CAN Message
                                                        Control Field 1 */
#define CAN_MBX1A *(volatile unsigned int *)0x720C     /* CAN 2 of 8 Bytes of
                                                        Mailbox 1 */
#define CAN_MBX1B *(volatile unsigned int *)0x720D     /* CAN 2 of 8 Bytes of
                                                        Mailbox 1 */
#define CAN_MBX1C *(volatile unsigned int *)0x720E     /* CAN 2 of 8 Bytes of
                                                        Mailbox 1 */
#define CAN_MBX1D *(volatile unsigned int *)0x720F     /* CAN 2 of 8 Bytes of
                                                        Mailbox 1 */
#define CAN_MSGID2L *(volatile unsigned int *)0x7210   /* CAN Message ID for
                                                        MB2 , low */
#define CAN_MSGID2H *(volatile unsigned int *)0x7211   /* CAN Message ID for
                                                        MB2 , high */
#define CAN_MSGCTRL2 *(volatile unsigned int *)0x7212 /* CAN Message Control
                                                        Field 2 */
#define CAN_MBX2A *(volatile unsigned int *)0x7214     /* CAN 2 of 8 Bytes of
                                                        Mailbox 2 */
#define CAN_MBX2B *(volatile unsigned int *)0x7215     /* CAN 2 of 8 Bytes of
                                                        Mailbox 2 */
#define CAN_MBX2C *(volatile unsigned int *)0x7216     /* CAN 2 of 8 Bytes of
                                                        Mailbox 2 */
#define CAN_MBX2D *(volatile unsigned int *)0x7217     /* CAN 2 of 8 Bytes of
                                                        Mailbox 2 */
#define CAN_MSGID3L *(volatile unsigned int *)0x7218   /* CAN Message ID for
                                                        MB3 , low */
#define CAN_MSGID3H *(volatile unsigned int *)0x7219   /* CAN Message ID for
                                                        MB3 , high */
#define CAN_MSGCTRL3 *(volatile unsigned int *)0x721A  /* CAN Message Control
                                                        Field 3 */
#define CAN_MBX3A *(volatile unsigned int *)0x721C   /* CAN 2 of 8 Bytes of
                                                        Mailbox 3 */
```

```
#define CAN_MBX3B *(volatile unsigned int *)0x721D     /* CAN 2 of 8 Bytes of
                                                              Mailbox 3 */
#define CAN_MBX3C  *(volatile unsigned int *)0x721E    /* CAN 2 of 8 Bytes of
                                                              Mailbox 3 */
#define CAN_MBX3D *(volatile unsigned int *)0x721F    /* CAN 2 of 8 Bytes of
                                                              Mailbox 3 */
#define CAN_MSGID4L *(volatile unsigned int *)0x7220   /* CAN Message ID
                                                          for MB4 , low */
#define CAN_MSGID4H *(volatile unsigned int *)0x7221 /* CAN Message ID for
                                                          MB4 , high */
#define CAN_MSGCTRL4 *(volatile unsigned int *)0x7222  /* CAN Message
                                                          Control Field 4 */
#define CAN_MBX4A     *(volatile unsigned int *)0x7224      /* CAN 2 of 8 Bytes of
                                                              Mailbox 4 */
#define CAN_MBX4B     *(volatile unsigned int *)0x7225      /* CAN 2 of 8 Bytes of
                                                              Mailbox 4 */
#define CAN_MBX4C     *(volatile unsigned int *)0x7226      /* CAN 2 of 8 Bytes
                                                          of Mailbox 4 */
#define CAN_MBX4D     *(volatile unsigned int *)0x7227      /* CAN 2 of 8 Bytes of
                                                              Mailbox 4 */
#define CAN_MSGID5L      *(volatile unsigned int *)0x7228      /* CAN Message ID
                                                          for MB5 , low */
#define CAN_MSGID5H      *(volatile unsigned int *)0x7229      /* CAN Message ID
                                                          for MB5 , high */
#define CAN_MSGCTRL5   *(volatile unsigned int *)0x722A      /* CAN Message
                                                          Control Field 5 */
#define CAN_MBX5A        *(volatile unsigned int *)0x722C      /* CAN 2 of 8 Bytes
                                                          of Mailbox 5 */
#define CAN_MBX5B        *(volatile unsigned int *)0x722D      /* CAN 2 of 8 Bytes
                                                          of Mailbox 5 */
#define CAN_MBX5C        *(volatile unsigned int *)0x722E      /* CAN 2 of 8 Bytes
                                                          of Mailbox 5 */
#define CAN_MBX5D        *(volatile unsigned int *)0x722F      /* CAN 2 of 8 Bytes
                                                          of Mailbox 5 */
#define GPTCON    *(volatile unsigned int *)0x7400   /* GP Timer Control  Register */
#define T1CNT    *(volatile unsigned int *)0x7401    /* GP Timer 1 Counter Register */
#define T1CMPR *(volatile unsigned int *)0x7402  /* GP Timer 1 Compare Register */
#define T1PR        *(volatile unsigned int *)0x7403    /* GP Timer 1 Period Register */
#define T1CON    *(volatile unsigned int *)0x7404    /* GP Timer 1 Control Register */
#define T2CNT    *(volatile unsigned int *)0x7405    /* GP Timer 2 Counter Register */
#define T2CMPR *(volatile unsigned int *)0x7406  /* GP Timer 2 Compare Register*/
#define T2PR      *(volatile unsigned int *)0x7407    /* GP Timer 2 Period Register */
#define T2CON   *(volatile unsigned int *)0x7408    /* GP Timer 2 Control Register */
#define COMCON   *(volatile unsigned int *)0x7411    /* Compare Control Register */
#define ACTR    *(volatile unsigned int *)0x7413    /* Full-Compare Action Register */
#define DBTCON   *(volatile unsigned int *)0x7415   /* Dead-Band Timer Control
                                                          Register */
#define CMPR1     *(volatile unsigned int *)0x7417  /* Full Compare Unit Compare
                                                          Register 1 */
```

```
#define CMPR2      *(volatile unsigned int *)0x7418    /* Full Compare Unit Compare
                                                          Register 2 */
#define CMPR3      *(volatile unsigned int *)0x7419    /* Full Compare Unit Compare
                                                          Register 3 */
#define CAPCON     *(volatile unsigned int *)0x7420     /* Capture Control Register */
#define CAPFIFO    *(volatile unsigned int *)0x7422 /* Capture FIFO Status Register */
#define CAP1FIFO    *(volatile unsigned int *)0x7423     /* Two-Level-Deep Capture
                                                          FIFO Stack 1 */
#define CAP2FIFO    *(volatile unsigned int *)0x7424     /* Two-Level-Deep Capture
                                                          FIFO Stack 2 */
#define CAP3FIFO    *(volatile unsigned int *)0x7425     /* Two-Level-Deep Capture
                                                          FIFO Stack 3 */
#define CAP1FBOT   *(volatile unsigned int *)0x7427     /* Capture 1 Bottom Stack
                                                          Register  */
#define CAP2FBOT   *(volatile unsigned int *)0x7428     /* Capture 2 Bottom Stack
                                                          Register */
#define CAP3FBOT   *(volatile unsigned int *)0x7429     /* Capture 3 Bottom Stack
                                                          Register */
#define EVIMRA *(volatile unsigned int *)0x742C  /* EV Interrupt Mask Register A */
#define EVIMRB *(volatile unsigned int *)0x742D  /* EV Interrupt Mask Register B */
#define EVIMRC *(volatile unsigned int *)0x742E  /* EV Interrupt Mask Register C */
#define EVIFRA  *(volatile unsigned int *)0x742F   /* EV Interrupt Flag Register A */
#define EVIFRB  *(volatile unsigned int *)0x7430    /* EV Interrupt Flag Register B */
#define EVIFRC  *(volatile unsigned int *)0x7431    /* EV Interrupt Flag Register C */
```

- Archivo GEL de inicio para Code Composer con registros de periféricos:

```
/*********************************************************/
/*                C24x General Purpose Timer Registers   */
/*********************************************************/
menuitem "Watch General Purpose Timer Registers";

hotmenu All_GP_Regs()
{
   GEL_WatchAdd("*0x7400,x","GPTCON");
   GEL_WatchAdd("*0x7401,x","T1CNT");
   GEL_WatchAdd("*0x7402,x","T1CMPR");
   GEL_WatchAdd("*0x7403,x","T1PR");
   GEL_WatchAdd("*0x7404,x","T1CON");
   GEL_WatchAdd("*0x7405,x","T2CNT");
   GEL_WatchAdd("*0x7406,x","T2CMPR");
   GEL_WatchAdd("*0x7407,x","T2PR");
   GEL_WatchAdd("*0x7408,x","T2CON");
}
hotmenu GPTCON()
{
   GEL_WatchAdd("*0x7400,x","GPTCON");
}
hotmenu T1CNT()
{
   GEL_WatchAdd("*0x7401,x","T1CNT");
}
hotmenu T1CMPR()
{
   GEL_WatchAdd("*0x7402,x","T1CMPR");
}
hotmenu T1PR()
{
   GEL_WatchAdd("*0x7403,x","T1PR");
}
hotmenu T1CON()
{
   GEL_WatchAdd("*0x7404,x","T1CON");
}
hotmenu T2CNT()
{
   GEL_WatchAdd("*0x7405,x","T2CNT");
}
hotmenu T2CMPR()
{
   GEL_WatchAdd("*0x7406,x","T2CMPR");
}
hotmenu T2PR()
{
```

```
   GEL_WatchAdd("*0x7407,x","T2PR");
}
hotmenu T2CON()
{
   GEL_WatchAdd("*0x7408,x","T2CON");
}

/*********************************************************/
/*                  C24x Compare Registers               */
/*********************************************************/
menuitem "Watch Compare Registers";

hotmenu All_CMP_Regs()
{
   GEL_WatchAdd("*0x7411,x","COMCON");
   GEL_WatchAdd("*0x7413,x","ACTR");
   GEL_WatchAdd("*0x7414,x","SACTR");
   GEL_WatchAdd("*0x7415,x","DBTCON");
   GEL_WatchAdd("*0x7417,x","CMPR1");
   GEL_WatchAdd("*0x7418,x","CMPR2");
   GEL_WatchAdd("*0x7419,x","CMPR3");
}
hotmenu COMCON()
{
   GEL_WatchAdd("*0x7411,x","COMCON");
}
hotmenu ACTR()
{
   GEL_WatchAdd("*0x7413,x","ACTR");
}
hotmenu SACTR()
{
   GEL_WatchAdd("*0x7414,x","SACTR");
}
hotmenu DBTCON()
{
   GEL_WatchAdd("*0x7415,x","DBTCON");
}
hotmenu CMPR1()
{
   GEL_WatchAdd("*0x7417,x","CMPR1");
}
hotmenu CMPR2()
{
   GEL_WatchAdd("*0x7418,x","CMPR2");
}
hotmenu CMPR3()
{
   GEL_WatchAdd("*0x7419,x","CMPR3");
}
```

```
/*********************************************************/
/*                    C24x Capture Registers             */
/*********************************************************/
menuitem "Watch Capture Registers";

hotmenu All_CAP_Regs()
{
   GEL_WatchAdd("*0x7420,x","CAPCON");
   GEL_WatchAdd("*0x7422,x","CAPFIFO");
   GEL_WatchAdd("*0x7423,x","CAP1FIFO");
   GEL_WatchAdd("*0x7424,x","CAP2FIFO");
   GEL_WatchAdd("*0x7425,x","CAP3FIFO");
   GEL_WatchAdd("*0x7427,x","CAP1FBOT");
   GEL_WatchAdd("*0x7428,x","CAP2FBOT");
   GEL_WatchAdd("*0x7429,x","CAP3FBOT");
}
hotmenu CAPCON()
{
   GEL_WatchAdd("*0x7420,x","CAPCON");
}
hotmenu CAPFIFO()
{
   GEL_WatchAdd("*0x7422,x","CAPFIFO");
}
hotmenu CAP1FIFO()
{
   GEL_WatchAdd("*0x7423,x","CAP1FIFO");
}
hotmenu CAP2FIFO()
{
   GEL_WatchAdd("*0x7424,x","CAP2FIFO");
}
hotmenu CAP3FIFO()
{
   GEL_WatchAdd("*0x7425,x","CAP3FIFO");
}
hotmenu CAP1FBOT()
{
   GEL_WatchAdd("*0x7427,x","CAP1FBOT");
}
hotmenu CAP2FBOT()
{
   GEL_WatchAdd("*0x7428,x","CAP2FBOT");
}
hotmenu CAP3FBOT()
{
   GEL_WatchAdd("*0x7429,x","CAP3FBOT");
}
```

```
/*********************************************************/
/*                C24x Event Manager Interrupt Registers */
/*********************************************************/
menuitem "Watch EvMgr Interrupt Registers";

hotmenu All_EVI_Regs()
{
   GEL_WatchAdd("*0x742c,x","EVIMRA");
   GEL_WatchAdd("*0x742d,x","EVIMRB");
   GEL_WatchAdd("*0x742e,x","EVIMRC");
   GEL_WatchAdd("*0x742f,x","EVIFRA");
   GEL_WatchAdd("*0x7430,x","EVIFRB");
   GEL_WatchAdd("*0x7431,x","EVIFRC");
}
hotmenu EVIMRA()
{
   GEL_WatchAdd("*0x742c,x","EVIMRA");
}
hotmenu EVIMRB()
{
   GEL_WatchAdd("*0x742d,x","EVIMRB");
}
hotmenu EVIMRC()
{
   GEL_WatchAdd("*0x742e,x","EVIMRC");
}
hotmenu EVIFRA()
{
   GEL_WatchAdd("*0x742f,x","EVIFRA");
}
hotmenu EVIFRB()
{
   GEL_WatchAdd("*0x7430,x","EVIFRB");
}
hotmenu EVIFRC()
{
   GEL_WatchAdd("*0x7431,x","EVIFRC");
}

/*********************************************************/
/*                  C24x A/D Converter Registers         */
/*********************************************************/
menuitem "Watch ADC Registers";

hotmenu All_ADC_Regs()
{
   GEL_WatchAdd("*0x7032,x","ADCTRL1");
   GEL_WatchAdd("*0x7034,x","ADCTRL2");
   GEL_WatchAdd("*0x7036,x","ADCFIFO1");
```

```
   GEL_WatchAdd("*0x7038,x","ADCFIFO2");
}
hotmenu ADCTRL1()
{
   GEL_WatchAdd("*0x7032,x","ADCTRL1");
}
hotmenu ADCTRL2()
{
   GEL_WatchAdd("*0x7034,x","ADCTRL2");
}
hotmenu ADCFIFO1()
{
   GEL_WatchAdd("*0x7036,x","ADCFIFO1");
}
hotmenu ADCFIFO2()
{
   GEL_WatchAdd("*0x7038,x","ADCFIFO2");
}

/*********************************************************/
/*                C24x Serial Peripheral Interface Registers                */
/*********************************************************/
menuitem "Watch SPI Registers";

hotmenu All_SPI_Regs()
{
   GEL_WatchAdd("*0x7040,x","SPICCR");
   GEL_WatchAdd("*0x7041,x","SPICTL");
   GEL_WatchAdd("*0x7042,x","SPISTS");
   GEL_WatchAdd("*0x7044,x","SPIBRR");
   GEL_WatchAdd("*0x7046,x","SPIRXEMU");
   GEL_WatchAdd("*0x7047,x","SPIRXBUF");
   GEL_WatchAdd("*0x7048,x","SPITXBUF");
   GEL_WatchAdd("*0x7049,x","SPIDAT");
   GEL_WatchAdd("*0x704F,x","SPIPRI");
}
hotmenu SPICCR()
{
   GEL_WatchAdd("*0x7040,x","SPICCR");
}
hotmenu SPICTL()
{
   GEL_WatchAdd("*0x7041,x","SPICTL");
}
hotmenu SPISTS()
{
   GEL_WatchAdd("*0x7042,x","SPISTS");
}
hotmenu SPIBRR()
{
```

```
   GEL_WatchAdd("*0x7044,x","SPIBRR");
}
hotmenu SPIDAT()
{
   GEL_WatchAdd("*0x7049,x","SPIDAT");
}
hotmenu SPIPRI()
{
   GEL_WatchAdd("*0x704F,x","SPIPRI");
}
hotmenu SPIRXEMU()
{
   GEL_WatchAdd("*0x7046,x","SPIRXEMU");
}
hotmenu SPIRXBUF()
{
   GEL_WatchAdd("*0x7047,x","SPIRXBUF");
}
hotmenu SPITXBUF()
{
   GEL_WatchAdd("*0x7048,x","SPITXBUF");
}

/*********************************************************/
/*          C24x Serial Communication Interface Registers          */
/*********************************************************/
menuitem "Watch SCI Registers";

hotmenu All_SCI_Regs()
{
   GEL_WatchAdd("*0x7050,x","SCICCR");
   GEL_WatchAdd("*0x7051,x","SCICTL1");
   GEL_WatchAdd("*0x7052,x","SCIHBAUD");
   GEL_WatchAdd("*0x7053,x","SCILBAUD");
   GEL_WatchAdd("*0x7054,x","SCICTL2");
   GEL_WatchAdd("*0x7055,x","SCIRXST");
   GEL_WatchAdd("*0x7056,x","SCIRXEMU");
   GEL_WatchAdd("*0x7057,x","SCIRXBUF");
   GEL_WatchAdd("*0x7059,x","SCITXBUF");
   GEL_WatchAdd("*0x705F,x","SCIPRI");
}
hotmenu SCICCR()
{
   GEL_WatchAdd("*0x7050,x","SCICCR");
}
hotmenu SCICTL1()
{
   GEL_WatchAdd("*0x7051,x","SCICTL1");
}
hotmenu SCIHBAUD()
```

```
{
   GEL_WatchAdd("*0x7052,x","SCIHBAUD");
}
hotmenu SCILBAUD()
{
   GEL_WatchAdd("*0x7053,x","SCILBAUD");
}
hotmenu SCICTL2()
{
   GEL_WatchAdd("*0x7054,x","SCICTL2");
}
hotmenu SCIRXST()
{
   GEL_WatchAdd("*0x7055,x","SCIRXST");
}
hotmenu SCIRXEMU()
{
   GEL_WatchAdd("*0x7056,x","SCIRXEMU");
}
hotmenu SCIRXBUF()
{
   GEL_WatchAdd("*0x7057,x","SCIRXBUF");
}
hotmenu SCITXBUF()
{
   GEL_WatchAdd("*0x7059,x","SCITXBUF");
}
hotmenu SCIPRI()
{
   GEL_WatchAdd("*0x705F,x","SCIPRI");
}

/*********************************************************/
/*              C24x Watchdog and Real-Time Interrupt Registers          */
/*********************************************************/
menuitem "Watch WD, RTI, & System Registers";

hotmenu All_WD_RTI_Regs()
{
   GEL_WatchAdd("*0x7023,x","WDCNTR");
   GEL_WatchAdd("*0x7025,x","WDKEY");
   GEL_WatchAdd("*0x7029,x","WDCR");
   GEL_WatchAdd("*0x7018,x","SCSR");
   GEL_WatchAdd("*0x701C,x","DINR");
}
hotmenu WDCNTR()
{
   GEL_WatchAdd("*0x7023,x","WDCNTR");
}
hotmenu WDKEY()
```

```
{
   GEL_WatchAdd("*0x7025,x","WDKEY");
}
hotmenu WDCR()
{
   GEL_WatchAdd("*0x7029,x","WDCR");
}
hotmenu DINR()
{
   GEL_WatchAdd("*0x701C,x","DINR");
}
hotmenu SCSR()
{
   GEL_WatchAdd("*0x7018,x","SCSR");
}
/************************************************************/
/*                    C24x Digital I/O Registers            */
/************************************************************/
menuitem "Watch Digital I/O Registers";

hotmenu All_IO_Regs()
{
   GEL_WatchAdd("*0x7090,x","OCRA");
   GEL_WatchAdd("*0x7092,x","OCRB");
   GEL_WatchAdd("*0x7098,x","PADATDIR");
   GEL_WatchAdd("*0x709A,x","PBDATDIR");
   GEL_WatchAdd("*0x709C,x","PCDATDIR");
   GEL_WatchAdd("*0x709E,x","PDDATDIR");
}
hotmenu OCRA()
{
   GEL_WatchAdd("*0x7090,x","OCRA");
}
hotmenu OCRB()
{
   GEL_WatchAdd("*0x7092,x","OCRB");
}
hotmenu PADATDIR()
{
   GEL_WatchAdd("*0x7098,x","PADATDIR");
}
hotmenu PBDATDIR()
{
   GEL_WatchAdd("*0x709A,x","PBDATDIR");
}
hotmenu PCDATDIR()
{
   GEL_WatchAdd("*0x709C,x","PCDATDIR");
}
hotmenu PDDATDIR()
```

```
{
   GEL_WatchAdd("*0x709E,x","PDDATDIR");
}

/*********************************************************/
/*                C24x External & I/F Interrupt Registers                */
/*********************************************************/
menuitem "Watch External & I/F Interrupt Registers";

hotmenu All_XIF_Regs()
{
   GEL_WatchAdd("*0x7070,x","XINT1CR");
   GEL_WatchAdd("*0x7071,x","XINT2CR");
   GEL_WatchAdd("*0x7010,x","PIRQR0");
   GEL_WatchAdd("*0x7011,x","PIRQR1");
   GEL_WatchAdd("*0x7014,x","PIACKR0");
   GEL_WatchAdd("*0x7015,x","PIACKR1");
   GEL_WatchAdd("*0x7018,x","SCSR");
   GEL_WatchAdd("*0x701E,x","PIVR");
}
hotmenu XINT1()
{
   GEL_WatchAdd("*0x7070,x","XINT1CR");
}
hotmenu NMI()
{
   GEL_WatchAdd("*0x7071,x","XINT2CR");
}
hotmenu PIRQR0()
{
   GEL_WatchAdd("*0x7010,x","PIRQR0");
}
hotmenu PIRQR1()
{
   GEL_WatchAdd("*0x7011,x","PIRQR1");
}
hotmenu PIACKR0()
{
   GEL_WatchAdd("*0x7014,x","PIACKR0");
}
hotmenu PIACKR1()
{
   GEL_WatchAdd("*0x7015,x","PIACKR1");
}
hotmenu PIVR()
{
   GEL_WatchAdd("*0x701E,x","PIVR");
}
```

INSTRUCTION SET SUMMARY

Table 1. TMS320x24x Opcode Symbols

SYMBOL	DESCRIPTION
A	Address
ACC	Accumulator
ACCB	Accumulator buffer
ARx	Auxiliary register value (0–7)
BITx	4-bit field that specifies which bit to test for the BIT instruction
BMAR	Block-move address register
DBMR	Dynamic bit-manipulation register
I	Addressing-mode bit
II...II	Immediate operand value
INTM	Interrupt-mode flag bit
INTR#	Interrupt vector number
K	Constant
PREG	Product register
PROG	Program memory
RPTC	Repeat counter
SHF, SHFT	3/4-bit shift value
TC	Test-control bit
T P	Two bits used by the conditional execution instructions to represent the conditions TC, NTC, and BIO. T P Meaning 0 0 $\overline{\text{BIO}}$ low 0 1 TC = 1 1 0 TC = 0 1 1 None of the above conditions
TREGn	Temporary register n (n = 0, 1, or 2)
Z L V C	4-bit field representing the following conditions: Z: ACC = 0 L: ACC < 0 V: Overflow C: Carry A conditional instruction contains two of these 4-bit fields. The 4-LSB field of the instruction is a 4-bit mask field. A 1 in the corresponding mask bit indicates that the condition is being tested. The second 4-bit field (bits 4–7) indicates the state of the conditions designated by the mask bits as being tested. For example, to test for ACC ≥ 0, the Z and L fields are set while the V and C fields are not set. The next 4-bit field contains the state of the conditions to test. The Z field is set to indicate testing of the condition ACC = 0, and the L field is reset to indicate testing of the condition ACC ≥ 0. The conditions possible with these 8 bits are shown in the BCND and CC instructions. To determine if the conditions are met, the 4-LSB bit mask is ANDed with the conditions. If any bits are set, the conditions are met.

INSTRUCTION SET SUMMARY (continued)

Table 2. TMS320x24x Instruction Set Summary

x24x MNEMONIC	DESCRIPTION	WORDS/ CYCLES	OPCODE MSB LSB
ABS	Absolute value of accumulator	1/1	1011 1110 0000 0000
ADD	Add to accumulator with shift	1/1	0010 SHFT IADD RESS
	Add to high accumulator	1/1	0110 0001 IADD RESS
	Add to accumulator short immediate	1/1	1011 1000 KKKK KKKK
	Add to accumulator long immediate with shift	2/2	1011 1111 1001 SHFT
ADDC	Add to accumulator with carry	1/1	0110 0000 IADD RESS
ADDS	Add to low accumulator with sign extension suppressed	1/1	0110 0010 IADD RESS
ADDT	Add to accumulator with shift specified by T register	1/1	0110 0011 IADD RESS
ADRK	Add to auxiliary register short immediate	1/1	0111 1000 KKKK KKKK
AND	AND with accumulator	1/1	0110 1110 IADD RESS
	AND immediate with accumulator with shift	2/2	1011 1111 1011 SHFT 16-Bit Constant
	AND immediate with accumulator with shift of 16	2/2	1011 1110 1000 0001 16-Bit Constant
APAC	Add P register to accumulator	1/1	1011 1110 0000 0100
B	Branch unconditionally	2/4	0111 1001 IADD RESS Branch Address
BACC	Branch to address specified by accumulator	1/4	1011 1110 0010 0000
BANZ	Branch on auxiliary register not zero	2/4/2	0111 1011 IADD RESS Branch Address
BCND	Branch if TC bit ≠ 0	2/4/2	1110 0001 0000 0000 Branch Address
	Branch if TC bit = 0	2/4/2	1110 0010 0000 0000 Branch Address
	Branch on carry	2/4/2	1110 0011 0001 0001 Branch Address
	Branch if accumulator ≥ 0	2/4/2	1110 0011 1000 1100 Branch Address
	Branch if accumulator > 0	2/4/2	1110 0011 0000 0100 Branch Address
	Branch on I/O status low	2/4/3	1110 0000 0000 0000 Branch Address
	Branch if accumulator ≤ 0	2/4/2	1110 0011 1100 1100 Branch Address
	Branch if accumulator < 0	2/4/2	1110 0011 0100 0100 Branch Address
	Branch on no carry	2/4/2	1110 0011 0000 0001 Branch Address
	Branch if no overflow	2/4/2	1110 0011 0000 0010 Branch Address

INSTRUCTION SET SUMMARY (continued)

Table 2. TMS320x24x Instruction Set Summary (Continued)

x24x MNEMONIC	DESCRIPTION	WORDS/ CYCLES	OPCODE			
			MSB			LSB
BCND	Branch if accumulator ≠ 0	2/4/2	1110	0011	0000	1000
			Branch Address			
	Branch on overflow	2/4/2	1110	0011	0010	0010
			Branch Address			
	Branch if accumulator = 0	2/4/2	1110	0011	1000	1000
			Branch Address			
BIT	Test bit	1/1	0100	BITx	IADD	RESS
BITT	Test bit specified by TREG	1/1	0110	1111	IADD	RESS
BLDD†	Block move from data memory to data memory source immediate	2/3	1010	1000	IADD	RESS
			Branch Address			
	Block move from data memory to data memory destination immediate	2/3	1010	1001	IADD	RESS
			Branch Address			
BLPD	Block move from program memory to data memory	2/3	1010	0101	IADD	RESS
			Branch Address			
CALA	Call subroutine indirect	1/4	1011	1110	0011	0000
CALL	Call subroutine	2/4	0111	1010	IADD	RESS
			Routine Address			
CC	Conditional call subroutine	2/4/2	1110	10TP	ZLVC	ZLVC
			Routine Address			
CLRC	Configure block as data memory	1/1	1011	1110	0100	0100
	Enable interrupt	1/1	1011	1110	0100	0000
	Reset carry bit	1/1	1011	1110	0100	1110
	Reset overflow mode	1/1	1011	1110	0100	0010
	Reset sign-extension mode	1/1	1011	1110	0100	0110
	Reset test/control flag	1/1	1011	1110	0100	1010
	Reset external flag	1/1	1011	1110	0100	1100
CMPL	Complement accumulator	1/1	1011	1110	0000	0001
CMPR	Compare auxiliary register with auxiliary register AR0	1/1	1011	1111	0100	01CM
DMOV	Data move in data memory	1/1	0111	0111	IADD	RESS
IDLE	Idle until interrupt	1/1	1011	1110	0010	0010
IN	Input data from port	2/2	1010	1111	IADD	RESS
			16BIT	I/O	PORT	ADRS
INTR	Software-interrupt	1/4	1011	1110	011K	KKKK
LACC	Load accumulator with shift	1/1	0001	SHFT	IADD	RESS
	Load accumulator long immediate with shift	2/2	1011	1111	1000	SHFT
			16-Bit Constant			
	Zero low accumulator and load high accumulator	1/1	0110	1010	IADD	RESS

† In x24x devices, the BLDD instruction does not work with memory-mapped registers IMR, IFR, and GREG.

INSTRUCTION SET SUMMARY (continued)

Table 2. TMS320x24x Instruction Set Summary (Continued)

x24x MNEMONIC	DESCRIPTION	WORDS/ CYCLES	OPCODE MSB			LSB
LACL	Load accumulator immediate short	1/1	1011	1001	KKKK	KKKK
	Zero accumulator	1/1	1011	1001	0000	0000
	Zero low accumulator and load high accumulator	1/1	0110	1010	IADD	RESS
	Zero low accumulator and load low accumulator with no sign extension	1/1	0110	1001	IADD	RESS
LACT	Load accumulator with shift specified by T register	1/1	0110	1011	IADD	RESS
LAR	Load auxiliary register	1/2	0000	0ARx	IADD	RESS
	Load auxiliary register short immediate	1/2	1011	0ARx	KKKK	KKKK
	Load auxiliary register long immediate	2/2	1011	1111	0000	1ARx
			16-Bit Constant			
LDP	Load data-memory page pointer	1/2	0000	1101	IADD	RESS
	Load data-memory page pointer immediate	1/2	1011	110P	AGEP	OINT
LPH	Load high-P register	1/1	0111	0101	IADD	RESS
LST	Load status register ST0	1/2	0000	1110	IADD	RESS
	Load status register ST1	1/2	0000	1111	IADD	RESS
LT	Load TREG	1/1	0111	0011	IADD	RESS
LTA	Load TREG and accumulate previous product	1/1	0111	0000	IADD	RESS
LTD	Load TREG, accumulate previous product, and move data	1/1	0111	0010	IADD	RESS
LTP	Load TREG and store P register in accumulator	1/1	0111	0001	IADD	RESS
LTS	Load TREG and subtract previous product	1/1	0111	0100	IADD	RESS
MAC	Multiply and accumulate	2/3	1010	0010	IADD	RESS
			16-Bit Constant			
MACD	Multiply and accumulate with data move	2/3	1010	0011	IADD	RESS
			16-Bit Constant			
MAR	Load auxiliary register pointer	1/1	1000	1011	1000	1ARx
	Modify auxiliary register	1/1	1000	1011	IADD	RESS
MPY	Multiply (with TREG, store product in P register)	1/1	0101	0100	IADD	RESS
	Multiply immediate	1/1	110C	KKKK	KKKK	KKKK
MPYA	Multiply and accumulate previous product	1/1	0101	0000	IADD	RESS
MPYS	Multiply and subtract previous product	1/1	0101	0001	IADD	RESS
MPYU	Multiply unsigned	1/1	0101	0101	IADD	RESS
NEG	Negate accumulator	1/1	1011	1110	0000	0010
NMI	Nonmaskable interrupt	1/4	1011	1110	0101	0010
NOP	No operation	1/1	1000	1011	0000	0000
NORM	Normalize contents of accumulator	1/1	1010	0000	IADD	RESS
OR	OR with accumulator	1/1	0110	1101	IADD	RESS
	OR immediate with accumulator with shift	2/2	1011	1111	1100	SHFT
			16-Bit Constant			
	OR immediate with accumulator with shift of 16	2/2	1011	1110	1000	0010
			16-Bit Constant			
OUT	Output data to port	2/3	0000 16BIT	1100 I/O	IADD PORT	RESS ADRS
PAC	Load accumulator with P register	1/1	1011	1110	0000	0011

INSTRUCTION SET SUMMARY (continued)

Table 2. TMS320x24x Instruction Set Summary (Continued)

x24x MNEMONIC	DESCRIPTION	WORDS/ CYCLES	OPCODE			
			MSB			LSB
POP	Pop top of stack to low accumulator	1/1	1011	1110	0011	0010
POPD	Pop top of stack to data memory	1/1	1000	1010	IADD	RESS
PSHD	Push data-memory value onto stack	1/1	0111	0110	IADD	RESS
PUSH	Push low accumulator onto stack	1/1	1011	1110	0011	1100
RET	Return from subroutine	1/4	1110	1111	0000	0000
RETC	Conditional return from subroutine	1/4/2	1110	11TP	ZLVC	ZLVC
ROL	Rotate accumulator left	1/1	1011	1110	0000	1100
ROR	Rotate accumulator right	1/1	1011	1110	0000	1101
RPT	Repeat instruction as specified by data-memory value	1/1	0000	1011	IADD	RESS
	Repeat instruction as specified by immediate value	1/1	1011	1011	KKKK	KKKK
SACH	Store high accumulator with shift	1/1	1001	1SHF	IADD	RESS
SACL	Store low accumulator with shift	1/1	1001	0SHF	IADD	RESS
SAR	Store auxiliary register	1/1	1000	0ARx	IADD	RESS
SBRK	Subtract from auxiliary register short immediate	1/1	0111	1100	KKKK	KKKK
SETC	Set carry bit	1/1	1011	1110	0100	1111
	Configure block as program memory	1/1	1011	1110	0100	0101
	Disable interrupt	1/1	1011	1110	0100	0001
	Set overflow mode	1/1	1011	1110	0100	0011
	Set test/control flag	1/1	1011	1110	0100	1011
	Set external flag XF	1/1	1011	1110	0100	1101
	Set sign-extension mode	1/1	1011	1110	0100	0111
SFL	Shift accumulator left	1/1	1011	1110	0000	1001
SFR	Shift accumulator right	1/1	1011	1110	0000	1010
SPAC	Subtract P register from accumulator	1/1	1011	1110	0000	0101
SPH	Store high-P register	1/1	1000	1101	IADD	RESS
SPL	Store low-P register	1/1	1000	1100	IADD	RESS
SPM	Set P register output shift mode	1/1	1011	1111	IADD	RESS
SQRA	Square and accumulate	1/1	0101	0010	IADD	RESS
SQRS	Square and subtract previous product from accumulator	1/1	0101	0011	IADD	RESS
SST	Store status register ST0	1/1	1000	1110	IADD	RESS
	Store status register ST1	1/1	1000	1111	IADD	RESS
SPLK	Store long immediate to data memory	2/2	1010	1110	IADD	RESS
			16-Bit Constant			
SUB	Subtract from accumulator long immediate with shift	2/2	1011	1111	1010	SHFT
			16-Bit Constant			
	Subtract from accumulator with shift	1/1	0011	SHFT	IADD	RESS
	Subtract from high accumulator	1/1	0110	0101	IADD	RESS
	Subtract from accumulator short immediate	1/1	1011	1010	KKKK	KKKK

INSTRUCTION SET SUMMARY (continued)

Table 2. TMS320x24x Instruction Set Summary (Continued)

x24x MNEMONIC	DESCRIPTION	WORDS/ CYCLES	OPCODE MSB			LSB
SUBB	Subtract from accumulator with borrow	1/1	0110	0100	IADD	RESS
SUBC	Conditional subtract	1/1	0000	1010	IADD	RESS
SUBS	Subtract from low accumulator with sign extension suppressed	1/1	0110	0110	IADD	RESS
SUBT	Subtract from accumulator with shift specified by TREG	1/1	0110	0111	IADD	RESS
TBLR	Table read	1/3	1010	0110	IADD	RESS
TBLW	Table write	1/3	1010	0111	IADD	RESS
TRAP	Software interrupt	1/4	1011	1110	0101	0001
XOR	Exclusive-OR with accumulator	1/1	0110	1100	IADD	RESS
	Exclusive-OR immediate with accumulator with shift	2/2	1011	1111	1101	SHFT
			16-Bit Constant			
	Exclusive-OR immediate with accumulator with shift of 16	2/2	1011	1110	1000	0011
			16-Bit Constant			
ZALR	Zero low accumulator and load high accumulator with rounding	1/1	0110	1000	IADD	RESS

development support

Texas Instruments offers an extensive line of development tools for the x24x generation of DSPs, including tools to evaluate the performance of the processors, generate code, develop algorithm implementations, and fully integrate and debug software and hardware modules.

The following products support development of x24x-based applications:

Software Development Tools:
Assembler/linker
Simulator
Optimizing ANSI C compiler
Application algorithms
C/Assembly debugger and code profiler

Hardware Development Tools:
Emulator XDS510™ (supports x24x multiprocessor system debug)

The *TMS320 DSP Development Support Reference Guide* (literature number SPRU011) contains information about development support products for all TMS320™ DSP family member devices, including documentation. Refer to this document for further information about TMS320™ DSP documentation or any other TMS320™ DSP support products from Texas Instruments. There is also an additional document, the *TMS320 Third-Party Support Reference Guide* (literature number SPRU052), which contains information from other companies in the industry about products related to the TMS320™ DSP. To receive copies of TMS320™ DSP literature, contact the Literature Response Center at 800/477-8924.

See Table 21 and Table 22 for complete listings of development support tools for the x24x. For information on pricing and availability, contact the nearest TI field sales office or authorized distributor.

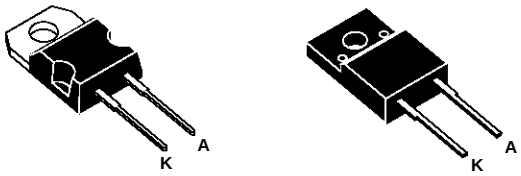
SGS-THOMSON MICROELECTRONICS

STTA506D/F

TURBOSWITCH ™ "A". ULTRA-FAST HIGH VOLTAGE DIODE

MAIN PRODUCTS CHARACTERISTICS

$I_{F(AV)}$	**5A**
V_{RRM}	**600V**
t_{rr} (typ)	**20ns**
V_F (max)	**1.5V**



TO220AC
(Plastic)

STTA506D

Isolated
ISOWATT220AC
(Plastic)

STTA506F

FEATURES AND BENEFITS

- SPECIFIC TO "FREEWHEEL MODE" OPERATIONS: Freewheel or Booster Diode.
- ULTRA-FAST RECOVERY.
- VERY LOW OVERALL POWER LOSSES IN BOTH THE DIODE AND THE COMPANION TRANSISTOR.
- HIGH FREQUENCY OPERATIONS.
- CECC APPROVED.

DESCRIPTION

The TURBOSWITCH is a very high performance series of ultra-fast high voltage power diodes from 600V to 1200V.
TURBOSWITCH, A family, drastically cuts losses in both the diode and the associated switching IGBT or MOSFET in all "Freewheel Mode" operations and is particularly suitable and efficient in Motor Control Freewheel applications and in Booster diode applications in Power Factor Control circuitries.
Packaged in TO220AC and in isolated ISOWATT220AC, these 600V devices are particularly intended for use on 240V domestic mains.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_{RRM}	Repetitive peak reverse voltage	600	V
V_{RSM}	Non repetitive peak reverse voltage	600	V
$I_{F(RMS)}$	RMS forward current	20	A
I_{FRM}	Repetitive peak forward current (tp = 5 µs, f = 5kHz)	65	A
T_j	Max operating junction temperature	-65 to 150	°C
T_{stg}	Storage temperature	-65 to 150	°C

TM : TURBOSWITCH is a trademark of **SGS-THOMSON MICROELECTRONICS.**

THERMAL AND POWER DATA

Symbol	Parameter	Conditions	Value	Unit
$R_{th(j-c)}$	Junction to case thermal resistance	STTA506D STTA506F	3.5 6.0	°C/W
P_1	Conduction power dissipation (see fig. 2)	$I_{F(AV)}$ = 5A δ =0.5 STTA506D Tc= 118°C STTA506F Tc= 96°C	9	W
P_{max}	Total power dissipation Pmax = P1 + P3 (P3 = 10% P1)	STTA506D Tc= 115°C STTA506F Tc= 90°C	10	W

STATIC ELECTRICAL CHARACTERISTICS (see Fig.2)

Symbol	Parameter	Test Conditions		Min	Typ	Max	Unit
V_F *	Forward voltage drop	I_F =5A	Tj = 25°C Tj = 125°C			1.75 1.5	V V
I_R **	Reverse leakage current	V_R =0.8 x V_{RRM}	Tj = 25°C Tj = 125°C			100 2	µA mA

Test pulses widths : * tp = 380 µs, duty cycle < 2%
** tp = 5 ms , duty cycle < 2%

DYNAMIC ELECTRICAL CHARACTERISTICS

TURN-OFF SWITCHING (see Fig.3)

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
t_{rr}	Reverse recovery time	Tj = 25°C I_F = 0.5 A I_R = 1A Irr = 0.25A I_F = 1 A dI_F/dt =-50A/µs V_R =30V		20	50	ns
I_{RM}	Maximum reverse recovery current	Tj = 125°C VR = 400V I_F =5A dI_F/dt = -40 A/µs dI_F/dt = -500 A/µs		11	3.0	A
S factor	Softness factor	Tj = 125°C V_R = 400V I_F =5A dI_F/dt = -500 A/µs		0.55		/

TURN-ON SWITCHING (see Fig.4)

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
t_{fr}	Forward recovery time	Tj = 25°C I_F =5 A, dI_F/dt = 40 A/µs measured at, 1.1 × $V_F$max			500	ns
V_{Fp}	Peak forward voltage	Tj = 25°C I_F =5A, dI_F/dt = 40 A/µs			10	V

APPLICATION DATA

The TURBOSWITCH "A" is especially designed to provide the lowest overall power losses in any "FREEWHEEL Mode" application (Fig.1) considering both the diode and the companion transistor, thus optimizing the overall performance in the end application.
The way of calculating the power losses is given below:

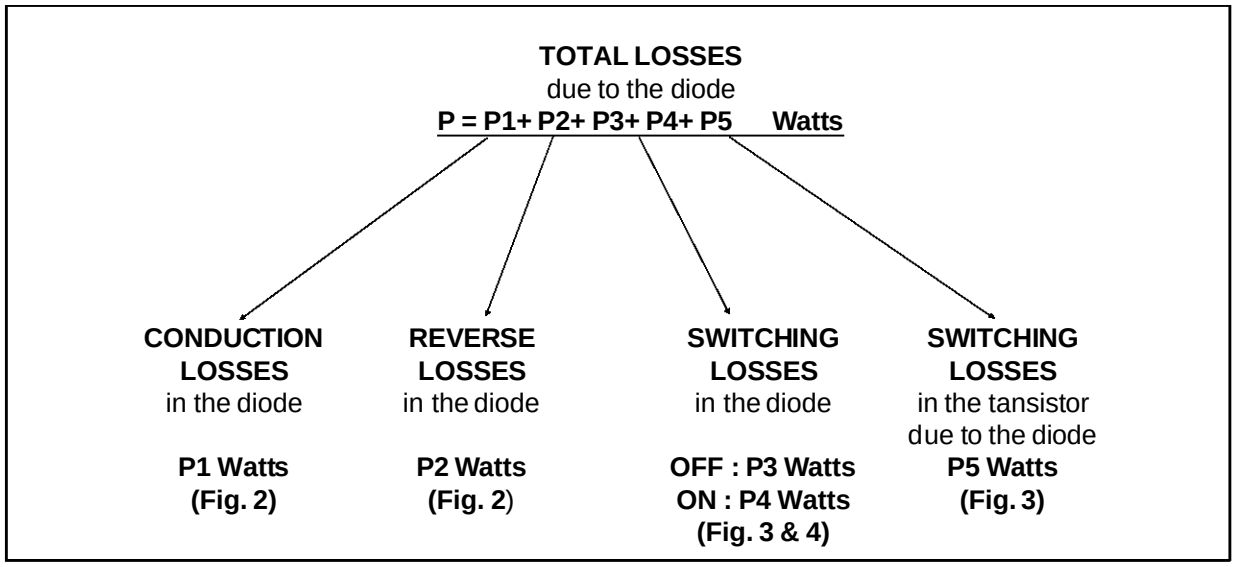
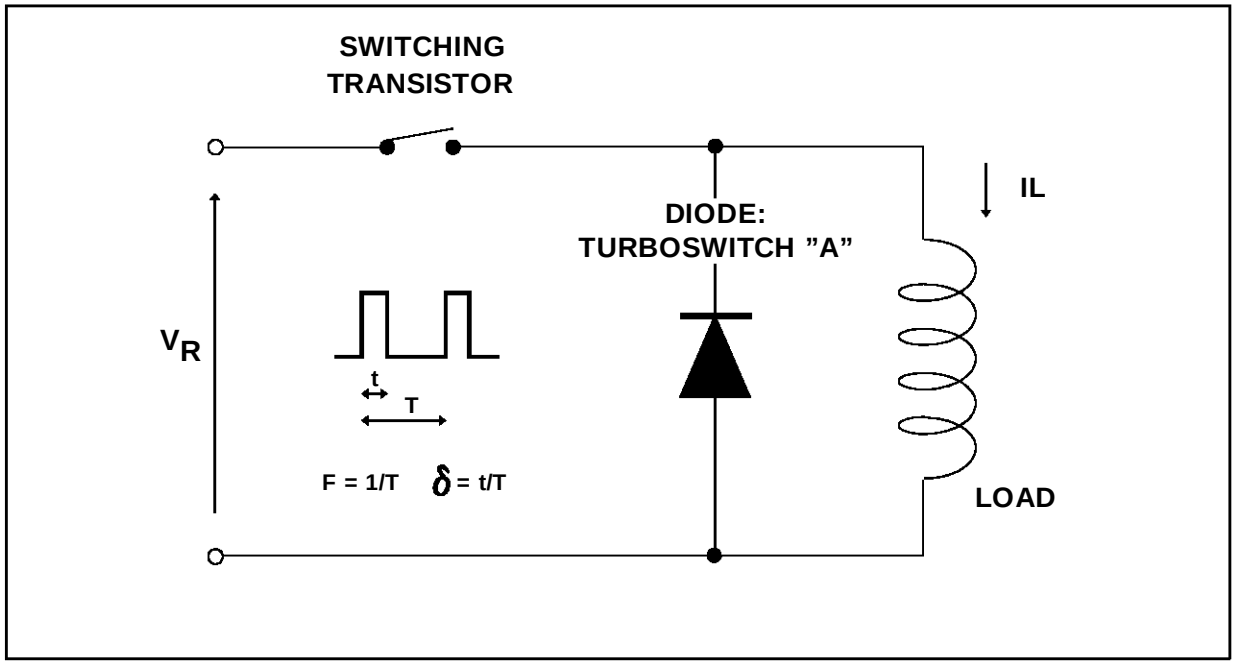
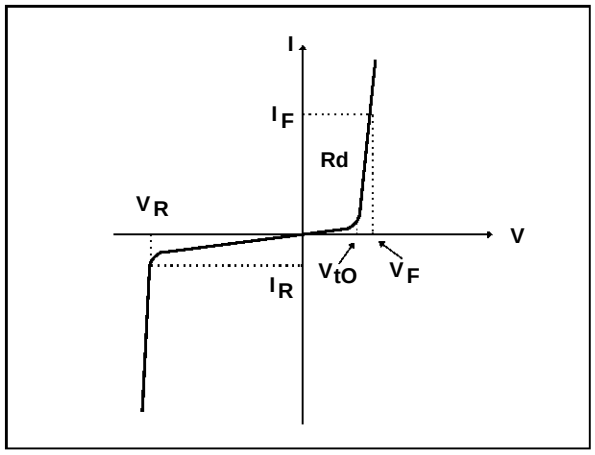


Fig. 1 : "FREEWHEEL" MODE.



APPLICATION DATA (Cont'd)

Fig. 2: STATIC CHARACTERISTICS



Conduction losses :

$P1 = V_{t0} \cdot I_{F(AV)} + R_d \cdot I_F{}^2{}_{(RMS)}$

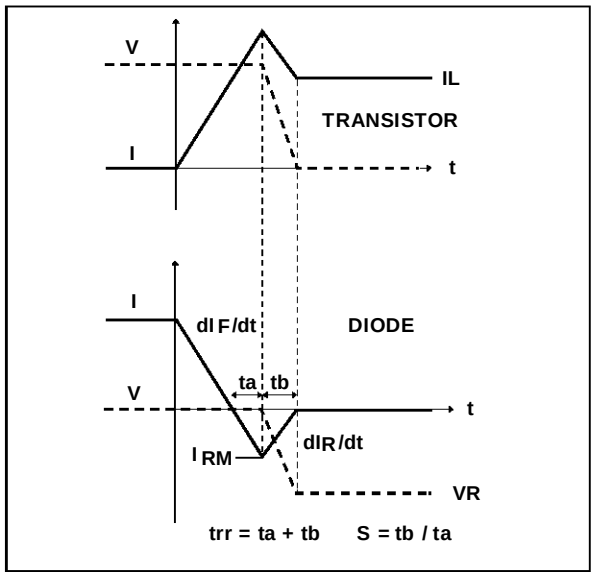
with

$V_{t0} = 1.15$ V
$R_d = 0.070$ Ohm
(Max values at 125°C)

Reverse losses :

$P2 = V_R \cdot I_R \cdot (1 - \delta)$

Fig. 3: TURN-OFF CHARACTERISTICS



Turn-on losses :
(in the transistor, due to the diode)

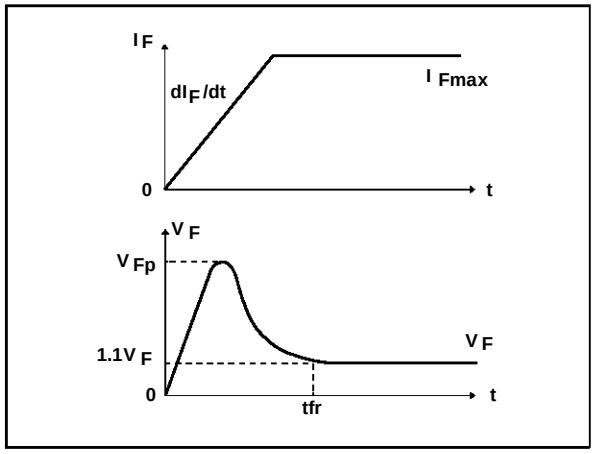
$$P5 = \frac{V_R \times I_{RM}{}^2 \times (3 + 2 \times S) \times F}{6 \; x \; dI_F/dt} + \frac{V_R \times I_{RM} \times I_L \times (S + 2) \times F}{2 \; x \; dI_F/dt}$$

Turn-off losses (in the diode) :

$$P3 = \frac{V_R \times I_{RM}{}^2 \times S \times F}{6 \; x \; dI_F/dt}$$

P3 and P5 are suitable for power MOSFET and IGBT

Fig. 4: TURN-ON CHARACTERISTICS



Turn-on losses :

$P4 = 0.4 \, (V_{FP} - V_F) \cdot I_{Fmax} \cdot t_{fr} \cdot F$

Fig 5 : Conduction losses versus average current

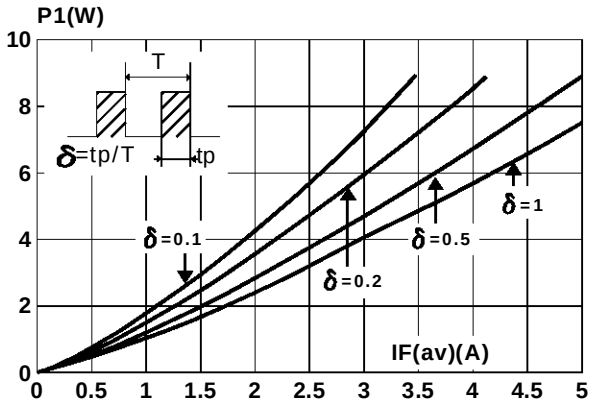


Fig 6 : Switching OFF losses versus dI_F/dt

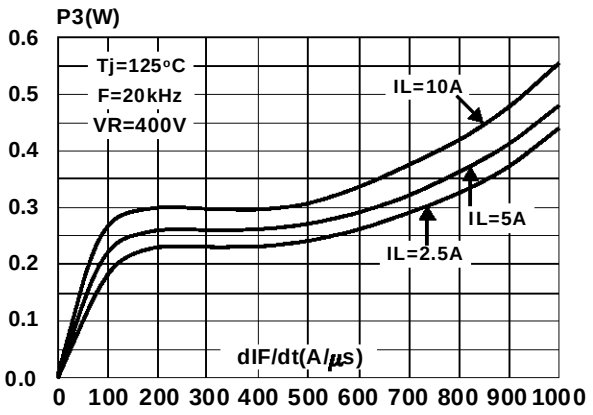


Fig 7 : Switching ON losses versus dI_F/dt

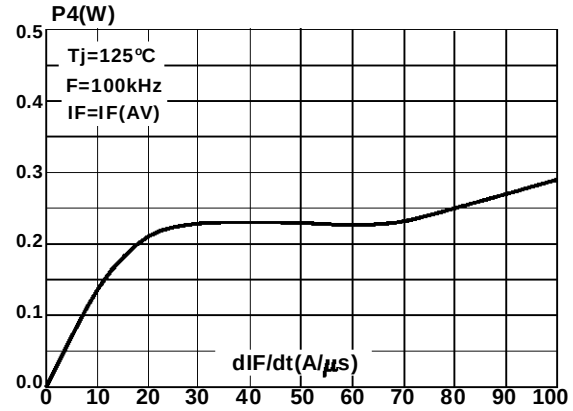


Fig 8 : Switching losses in transistor due to the diode

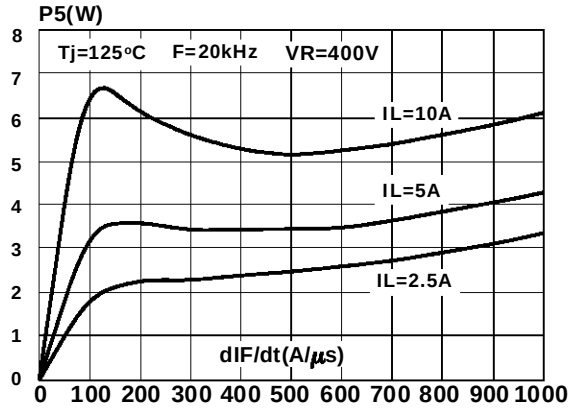


Fig 9 : Forward voltage drop versus forward current

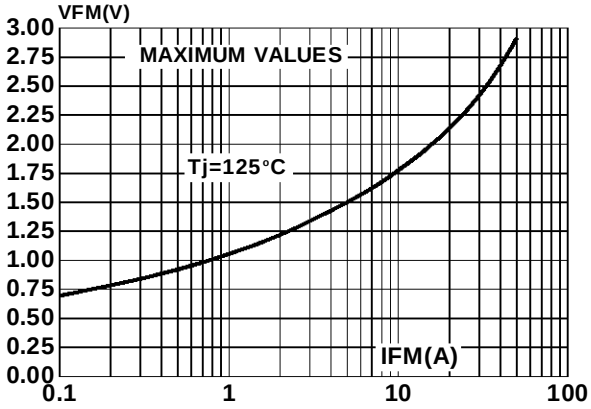


Fig 10 : Peak reverse recovery current versus dI_F/dt

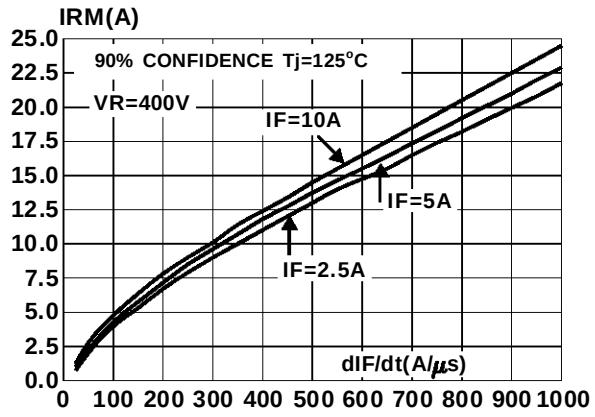


Fig 11 : Reverse recovery time versus dI_F/dt

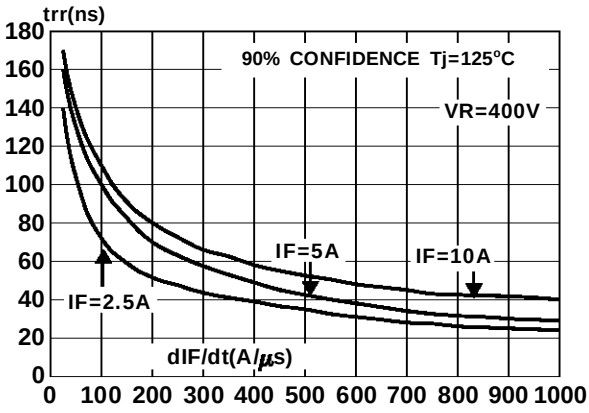


Fig 12 : Softness factor (tb/ta) versus dI_F/dt

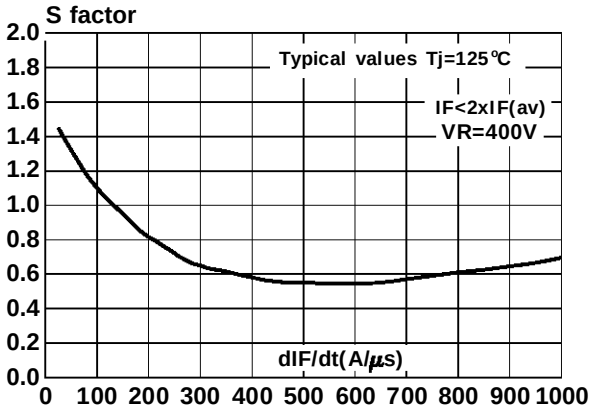


Fig 13 : Relative variation of dynamic parameters versus junction temperature (Reference Tj=125°c)

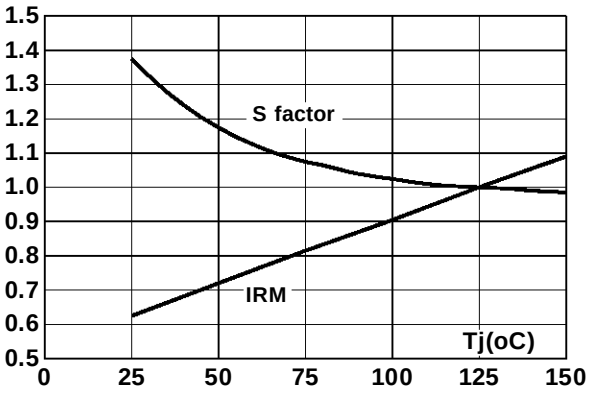


Fig 14 : Transient peak forward voltage versus dI_F/dt

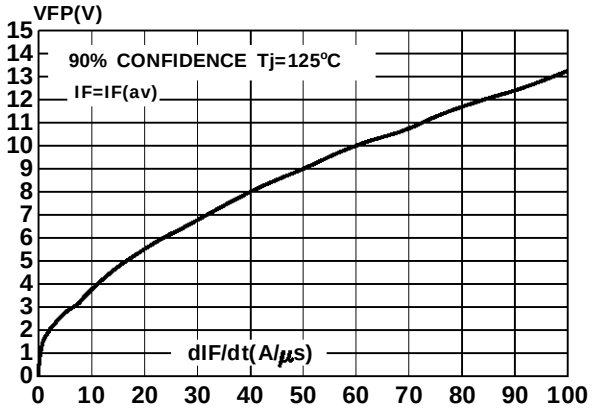


Fig 15 : Forward recovery time versus dI_F/dt

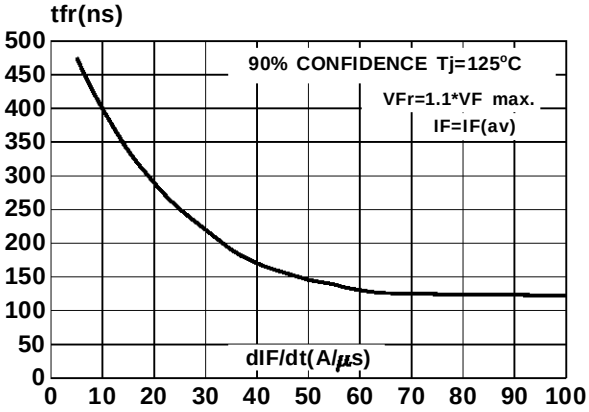


Fig 16 : Relative variation of thermal transient impedance junction to case versus pulse duration (TO220AC)

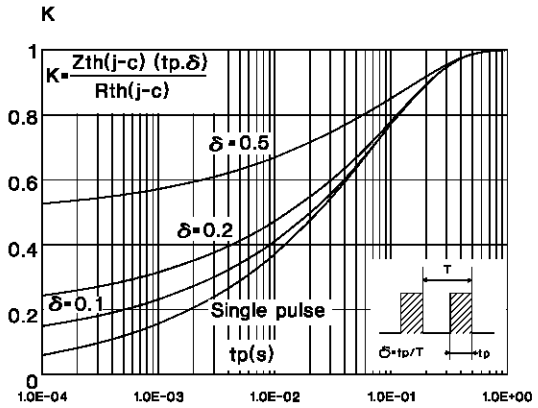
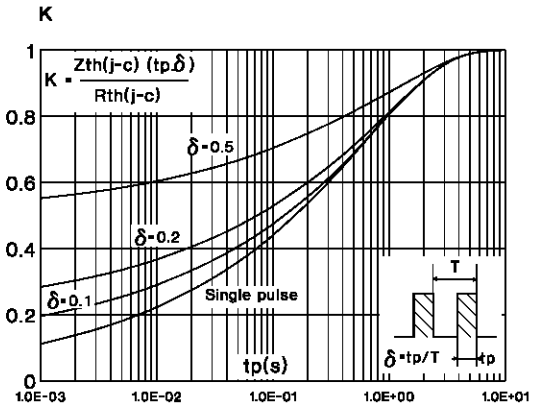
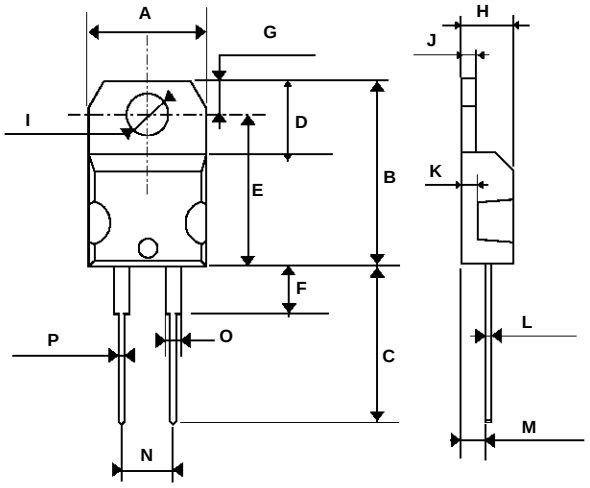


Fig 17 : Relative variation of thermal transient impedance junction to case versus pulse duration (ISOWATT220AC)



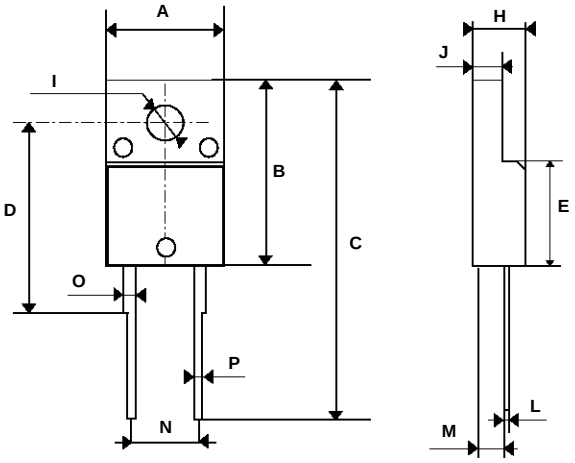
PACKAGE DATA
TO220AC (JEDEC outline)



REF.	DIMENSIONS			
	Millimeters		Inches	
	Min.	Max.	Min.	Max.
A	9.66	10.66	0.380	0.419
B	15.2	15.9	0.598	0.626
C	13	14	0.511	0.551
D	6.2	6.6•	0.244	0.260
E	16.4 typ.		0.645 typ.	
F	3.5	4.2	0.137	0.165
G	2.65	2.95	0.104	0.116
H	4.4	4.6	0.173	0.181
I	3.75	3.85	0.147	0.151
J	1.23	1.32	0.048	0.051
K	1.27 typ.		0.050 typ.	
L	0.49	0.70	0.019	0.027
M	2.4	2.72	0.094	0.107
N	4.95	5.15	0.194	0.203
O	1.14	1.70	0.044	0.067
P	0.61	0.88	0.024	0.034

Cooling method : C.
Marking : Type number.
Weight : 1.9 g.
Torque value : 0.55 m.N typ (0.70 m.N max).

PACKAGE DATA
ISOLATED ISOWATT220AC (JEDEC outline)



REF.	DIMENSIONS			
	Millimeters		Inches	
	Min.	Max.	Min.	Max.
A	10	10.4	0.393	0.409
B	15.9	16.4	0.626	0.645
C	28.6	30.6	1.126	1.204
D	16 typ		0.630 typ	
E	9	9.3	0.354	0.366
H	4.4	4.6	0.173	0.181
I	3	3.2	0.118	0.126
J	2.5	2.7	0.098	0.106
L	0.4	0.7	0.015	0.027
M	2.4	2.75	0.094	0.108
N	4.95	5.2	0.195	0.204
O	1.15	1.7	0.045	0.067
P	0.75	1	0.030	0.039

Cooling method : C.
Marking : Type number.
Weight : 2.0 g.
Torque value : 0.55 m.N typ (0.7 m.N max).

Electrical isolation : 2000V DC
Capacitance : 12 pF

Advanced Power MOSFET

IRF530A

FEATURES

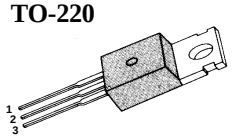
- Avalanche Rugged Technology
- Rugged Gate Oxide Technology
- Lower Input Capacitance
- Improved Gate Charge
- Extended Safe Operating Area
- 175℃ Operating Temperature
- Lower Leakage Current : 10 μA (Max.) @ V_{DS} = 100V
- Lower $R_{DS(ON)}$: 0.092 Ω(Typ.)

BV_{DSS} = 100 V

$R_{DS(on)}$ = 0.11Ω

I_D = 14 A

TO-220



1.Gate 2. Drain 3. Source

Absolute Maximum Ratings

Symbol	Characteristic	Value	Units
V_{DSS}	Drain-to-Source Voltage	100	V
I_D	Continuous Drain Current (T_C=25℃)	14	A
	Continuous Drain Current (T_C=100℃)	9.9	
I_{DM}	Drain Current-Pulsed ①	56	A
V_{GS}	Gate-to-Source Voltage	± 20	V
E_{AS}	Single Pulsed Avalanche Energy ②	261	mJ
I_{AR}	Avalanche Current ①	14	A
E_{AR}	Repetitive Avalanche Energy ①	5.5	mJ
dv/dt	Peak Diode Recovery dv/dt ③	6.5	V/ns
P_D	Total Power Dissipation (T_C=25℃)	55	W
	Linear Derating Factor	0.36	W/℃
T_J , T_{STG}	Operating Junction and Storage Temperature Range	- 55 to +175	℃
T_L	Maximum Lead Temp. for Soldering Purposes, 1/8" from case for 5-seconds	300	

Thermal Resistance

Symbol	Characteristic	Typ.	Max.	Units
$R_{\theta JC}$	Junction-to-Case	--	2.74	℃/W
$R_{\theta CS}$	Case-to-Sink	0.5	--	
$R_{\theta JA}$	Junction-to-Ambient	--	62.5	

FAIRCHILD SEMICONDUCTOR™

Electrical Characteristics (T_C=25℃ unless otherwise specified)

Symbol	Characteristic	Min.	Typ.	Max.	Units	Test Condition
BV_{DSS}	Drain-Source Breakdown Voltage	100	--	--	V	V_{GS}=0V,I_D=250μA
$\Delta BV/\Delta T_J$	Breakdown Voltage Temp. Coeff.	--	0.11	--	V/℃	I_D=250 μA ***See Fig 7***
$V_{GS(th)}$	Gate Threshold Voltage	2.0	--	4.0	V	V_{DS}=5V,I_D=250μA
I_{GSS}	Gate-Source Leakage , Forward	--	--	100	nA	V_{GS}=20V
	Gate-Source Leakage , Reverse	--	--	-100		V_{GS}=-20V
I_{DSS}	Drain-to-Source Leakage Current	--	--	10	μA	V_{DS}=100V
		--	--	100		V_{DS}=80V,T_C=150℃
$R_{DS(on)}$	Static Drain-Source On-State Resistance	--	--	0.11	Ω	V_{GS}=10V,I_D=7A ④
g_{fs}	Forward Transconductance	--	10.25	--	Ω	V_{DS}=40V,I_D=7A ④
C_{iss}	Input Capacitance	--	610	790	pF	V_{GS}=0V,V_{DS}=25V,f =1MHz ***See Fig 5***
C_{oss}	Output Capacitance	--	150	175		
C_{rss}	Reverse Transfer Capacitance	--	62	72		
$t_{d(on)}$	Turn-On Delay Time	--	13	40	ns	V_{DD}=50V,I_D=14A, R_G=12Ω ***See Fig 13*** ④⑤
t_r	Rise Time	--	14	40		
$t_{d(off)}$	Turn-Off Delay Time	--	55	110		
t_f	Fall Time	--	36	80		
Q_g	Total Gate Charge	--	27	36	nC	V_{DS}=80V,V_{GS}=10V, I_D=14A ***See Fig 6 & Fig 12*** ④⑤
Q_{gs}	Gate-Source Charge	--	4.5	--		
Q_{gd}	Gate-Drain(“Miller”) Charge	--	12.8	--		

Source-Drain Diode Ratings and Characteristics

Symbol	Characteristic	Min.	Typ.	Max.	Units	Test Condition
I_S	Continuous Source Current	--	--	14	A	Integral reverse pn-diode in the MOSFET
I_{SM}	Pulsed-Source Current ①	--	--	56		
V_{SD}	Diode Forward Voltage ④	--	--	1.5	V	T_J=25℃,I_S=14A,V_{GS}=0V
t_{rr}	Reverse Recovery Time	--	109	--	ns	T_J=25℃,I_F=14A
Q_{rr}	Reverse Recovery Charge	--	0.41	--	¥ìC	di_F/dt=100A/μs ④

Notes ;

① Repetitive Rating : Pulse Width Limited by Maximum Junction Temperature

② L=2mH, I_{AS}=14A, V_{DD}=25V, R_G=27Ω, Starting T_J=25°C

③ I_{SD} ≤ 14A, di/dt≤ 350A/μs, V_{DD}≤ BV_{DSS} , Starting T_J=25°C

④ Pulse Test : Pulse Width = 250 μs, Duty Cycle ≤2%

⑤ Essentially Independent of Operating Temperature

Fig 1. Output Characteristics

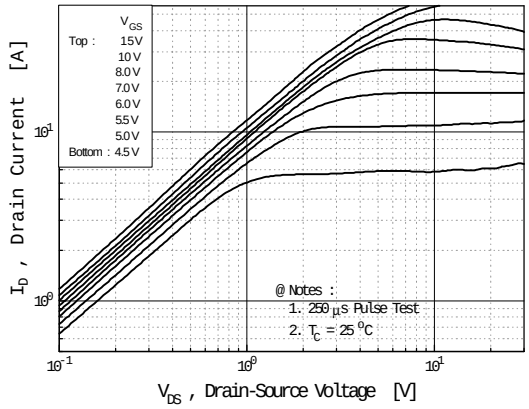


Fig 2. Transfer Characteristics

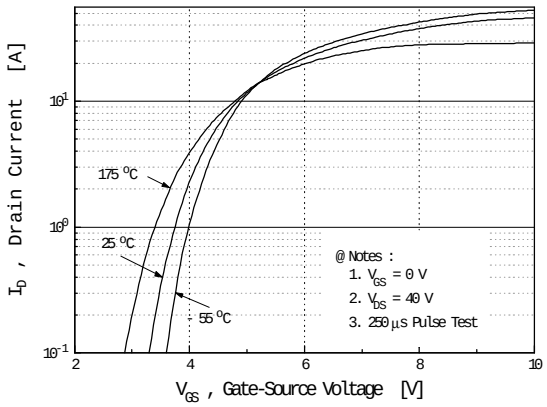


Fig 3. On-Resistance vs. Drain Current

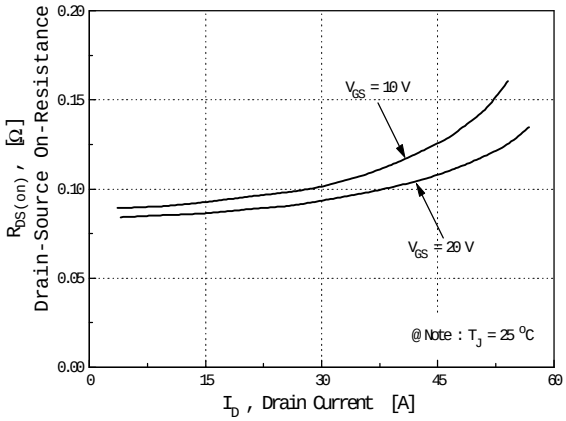


Fig 4. Source-Drain Diode Forward Voltage

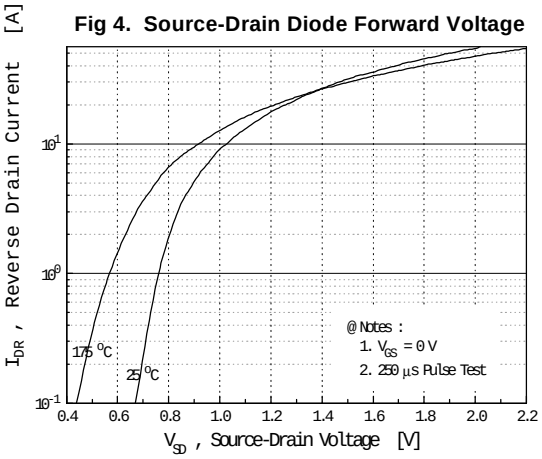


Fig 5. Capacitance vs. Drain-Source Voltage

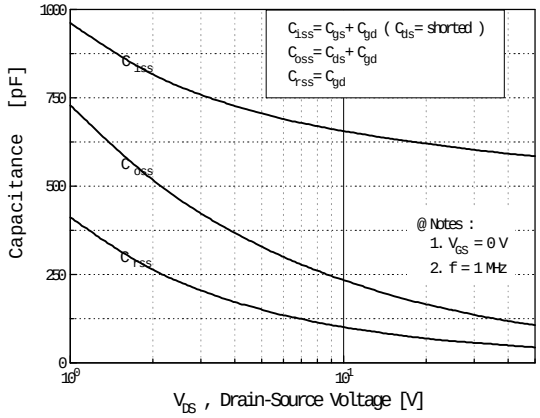


Fig 6. Gate Charge vs. Gate-Source Voltage

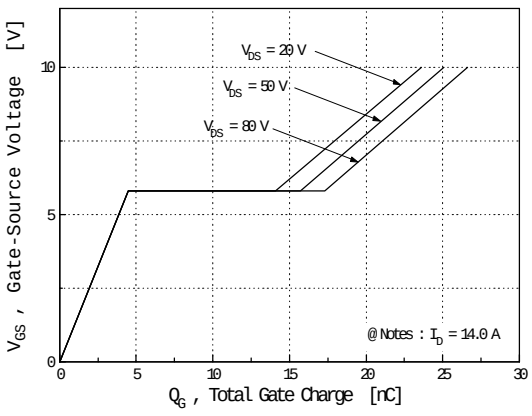


Fig 7. Breakdown Voltage vs. Temperature

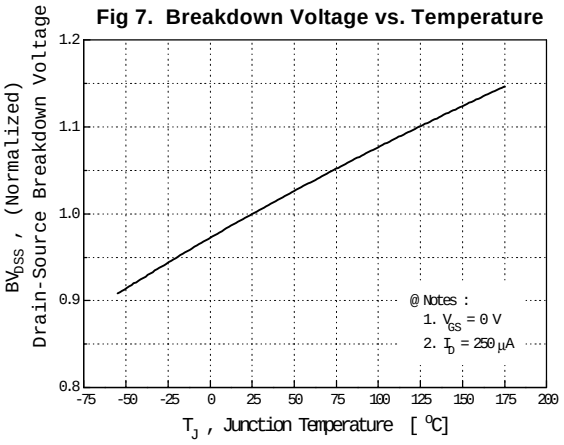


Fig 8. On-Resistance vs. Temperature

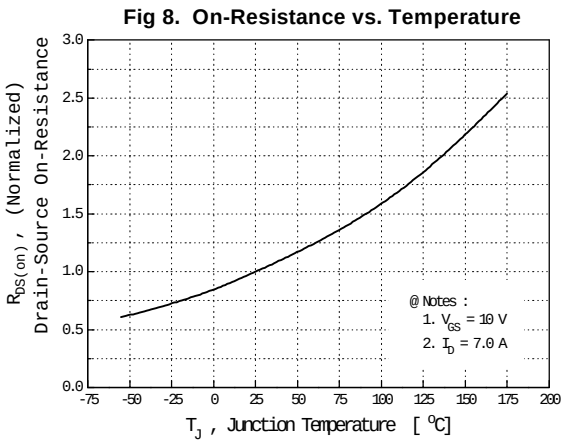


Fig 9. Max. Safe Operating Area

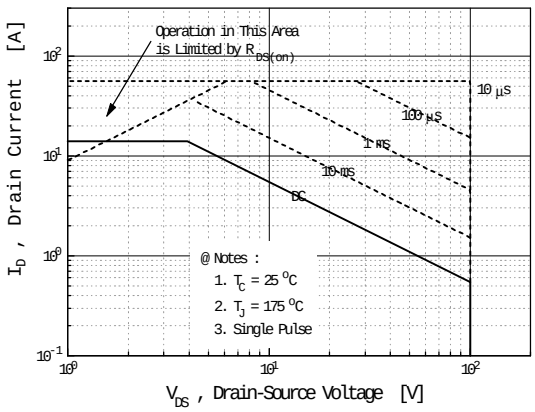


Fig 10. Max. Drain Current vs. Case Temperature

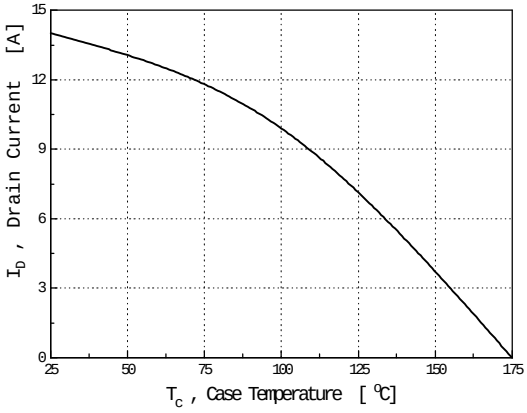


Fig 11. Thermal Response

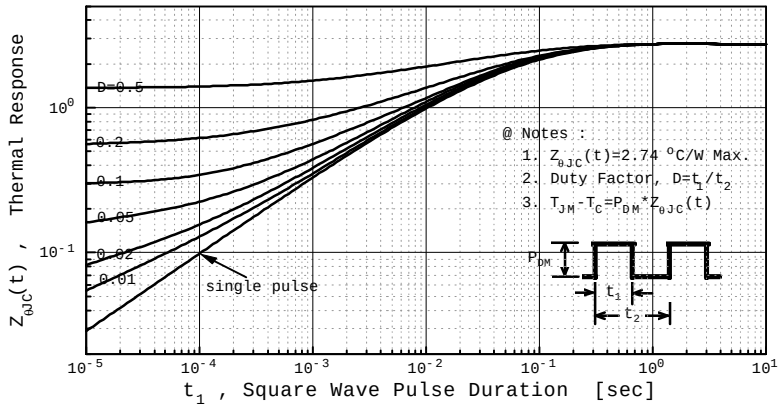


Fig 12. Gate Charge Test Circuit & Waveform

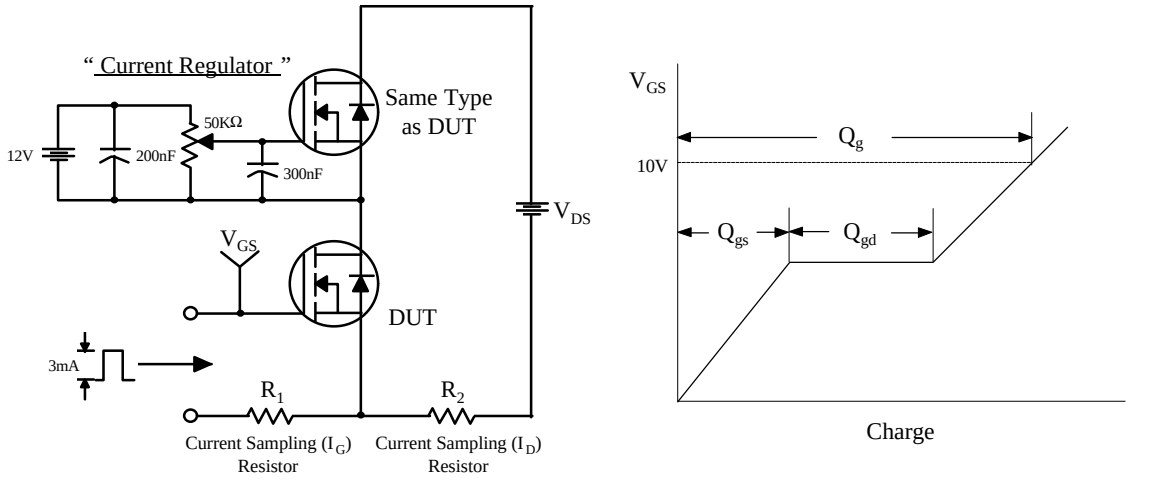


Fig 13. Resistive Switching Test Circuit & Waveforms

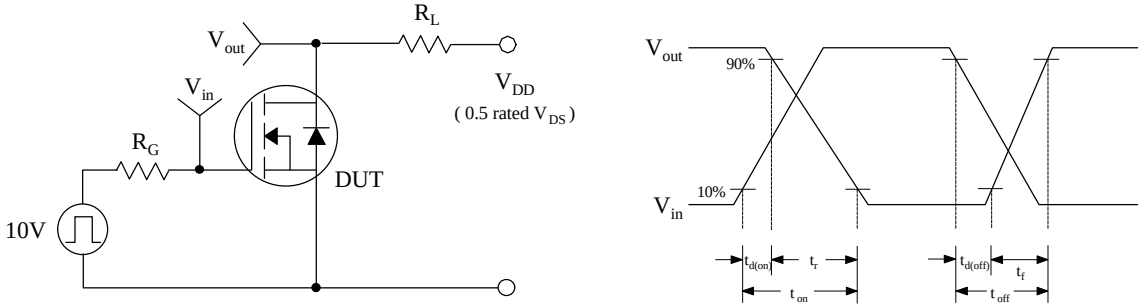


Fig 14. Unclamped Inductive Switching Test Circuit & Waveforms

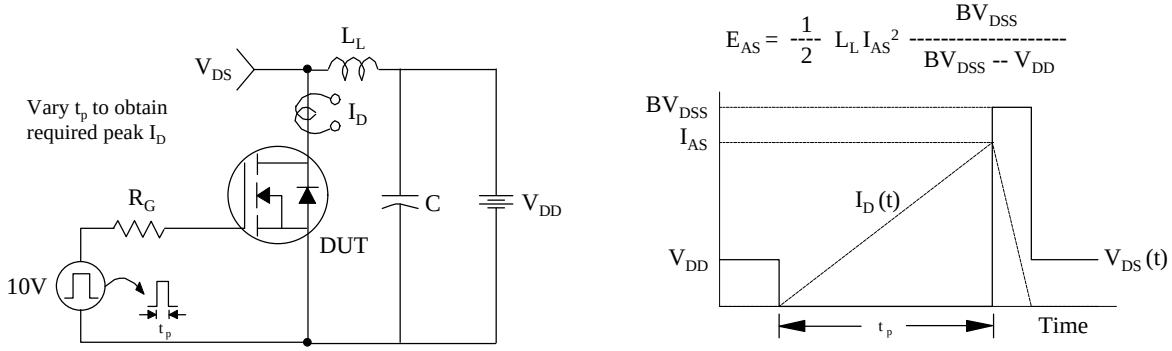
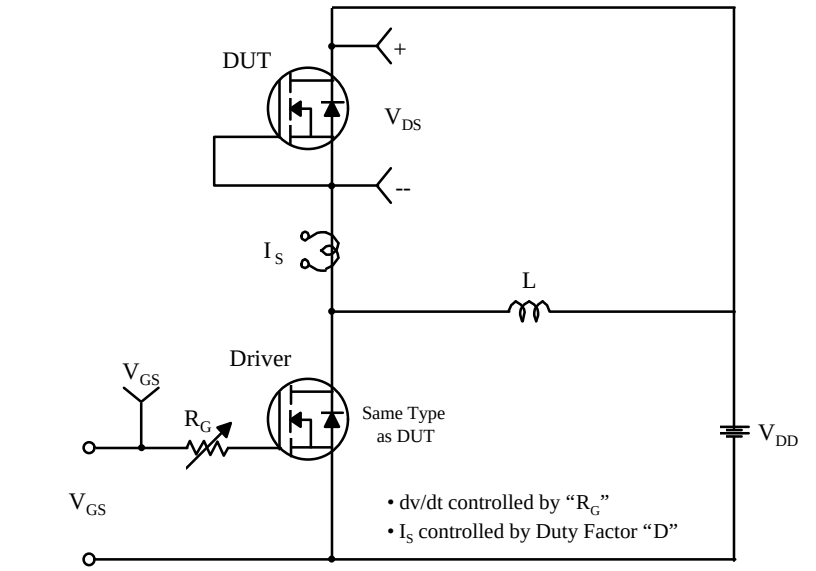
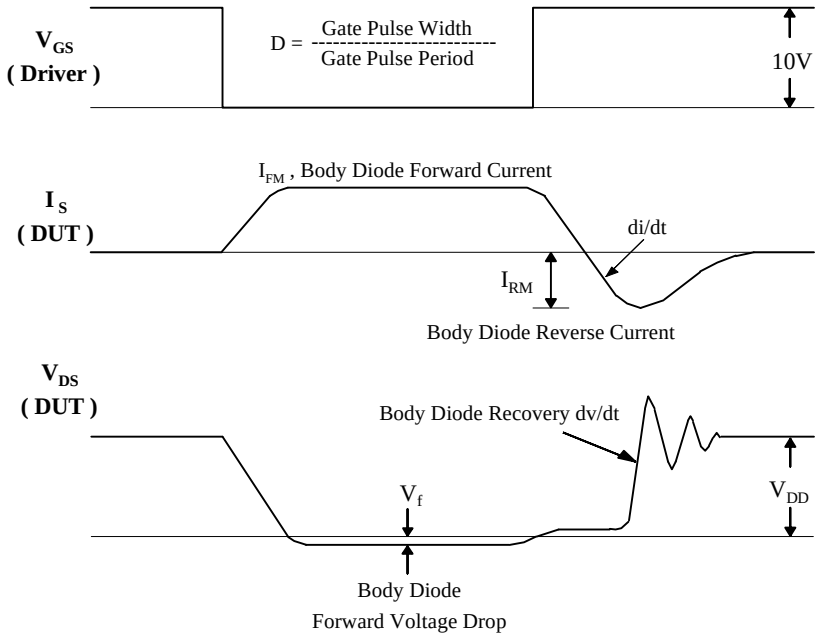


Fig 15. Peak Diode Recovery dv/dt Test Circuit & Waveforms





TRADEMARKS

The following are registered and unregistered trademarks Fairchild Semiconductor owns or is authorized to use and is not intended to be an exhaustive list of all such trademarks.

ACEx™
CoolFET™
CROSSVOLT™
E^2CMOS^{TM}
FACT™
FACT Quiet Series™
FAST®
FASTr™
GTO™
HiSeC™
ISOPLANAR™
MICROWIRE™
POP™
PowerTrench™
QS™
Quiet Series™
SuperSOT™-3
SuperSOT™-6
SuperSOT™-8
TinyLogic™
UHC™
VCX™

DISCLAIMER

FAIRCHILD SEMICONDUCTOR RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN TO IMPROVE RELIABILITY, FUNCTION OR DESIGN. FAIRCHILD DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT DESCRIBED HEREIN; NEITHER DOES IT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS, NOR THE RIGHTS OF OTHERS.

LIFE SUPPORT POLICY

FAIRCHILD'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF FAIRCHILD SEMICONDUCTOR CORPORATION.
As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, or (c) whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

PRODUCT STATUS DEFINITIONS

Definition of Terms

Datasheet Identification	Product Status	Definition
Advance Information	Formative or In Design	This datasheet contains the design specifications for product development. Specifications may change in any manner without notice.
Preliminary	First Production	This datasheet contains preliminary data, and supplementary data will be published at a later date. Fairchild Semiconductor reserves the right to make changes at any time without notice in order to improve design.
No Identification Needed	Full Production	This datasheet contains final specifications. Fairchild Semiconductor reserves the right to make changes at any time without notice in order to improve design.
Obsolete	Not In Production	This datasheet contains specifications on a product that has been discontinued by Fairchild semiconductor. The datasheet is printed for reference information only.

19-0132; Rev 1, 3/93

MAXIM

Dual High-Speed 1.5A MOSFET Drivers

General Description

The MAX4426/4427/4428 are dual monolithic MOSFET drivers designed to translate TTL/CMOS inputs to high voltage/current outputs. The MAX4426 is a dual inverting power MOSFET driver. The MAX4427 is a dual noninverting power MOSFET driver, and the MAX4428 contains one inverting section and one noninverting section. Delay times are nearly independent of V_{DD} (see *Typical Operating Characteristics*). High-current output drivers rapidly charge and discharge the gate capacitance of even the largest power MOSFETs to within millivolts of the supply rails. This produces the power MOSFETs' minimum on resistance. The MAX4426/4427/4428's high speed minimizes power losses in switching power supplies and DC-DC converters.

Applications

Switching Power Supplies

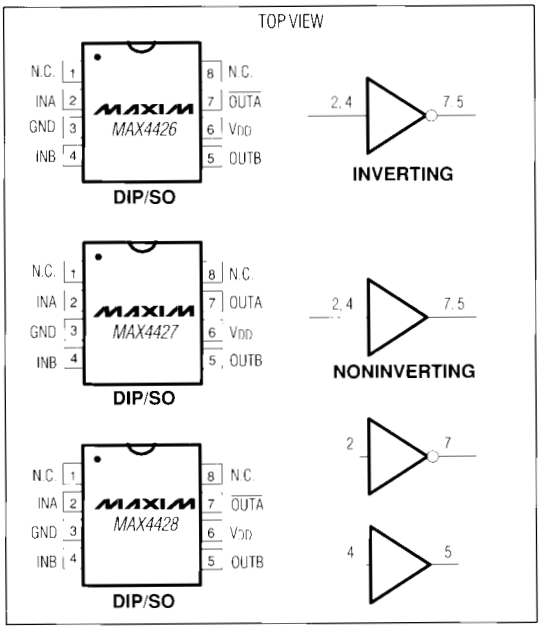
DC-DC Converters

Motor Controllers

Pin-Diode Drivers

Charge-Pump Voltage Inverters

Pin Configurations



Features

- ♦ **Upgrade for TSC4426/4427/4428**
- ♦ **Lower On Resistance: 4Ω vs. 7Ω**
- ♦ **Shorter Delay Times: t_{D1} – 10ns vs. 30ns**
  t_{D2} – 25ns vs. 50ns
- ♦ **1.5A Peak Output Current**
- ♦ **Fast Rise and Fall Times: Typically 20ns with 1000pF Load**
- ♦ **Wide Operating Range: 4.5V to 18V**
- ♦ **Low Power Consumption: 1.8mA with Logic 1 Input**
  200µA with Logic 0 Input
- ♦ **TTL/CMOS Compatible**
- ♦ **Latchup Protected - Withstand >500mA Reverse Current**
- ♦ **ESD Protected**

Ordering Information

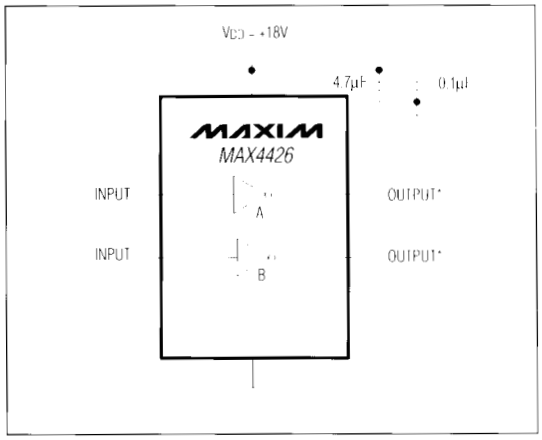
PART	TEMP. RANGE	PIN-PACKAGE
MAX4426CPA	0 C to +70 C	8 Plastic DIP
MAX4426CSA	0 C to +70 C	8 SO
MAX4426C/D	0 C to +70 C	Dice*
MAX4426EPA	-40 C to +85 C	8 Plastic DIP
MAX4426ESA	-40 C to +85 C	8 SO
MAX4426EJA	-40 C to +85 C	8 CERDIP
MAX4426MJA	-55 C to +125 C	8 CERDIP**

Ordering Information continued on last page.

* *Dice are tested at T_A = +25°C.*

** *Contact factory for availability and processing to MIL-STD-883.*

Typical Operating Circuit



Dual High-Speed 1.5A MOSFET Drivers

ABSOLUTE MAXIMUM RATINGS

Supply Voltage V_{DD} to GND ..... +20V
Input Voltage ..... V_{DD} + 0.3V to GND - 0.3V
Continuous Power Dissipation (T_A = +70°C)
Plastic DIP (derate 9.09mW/°C above +70°C) 727mW
SO (derate 5.88mW/°C above +70°C) 471mW
CERDIP (derate 8.00mW/°C above 70°C) 640mW

Operating Temperature Ranges:
MAX442_C_ ..... 0°C to +70°C
MAX442_E_ ..... -40°C to +85°C
MAX442_MJA ..... -55°C to +125°C
Storage Temperature Range ..... -55°C to +160°C
Maximum Chip Temperature ..... +150°C
Lead Temperature (soldering, 10 sec) ..... +300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

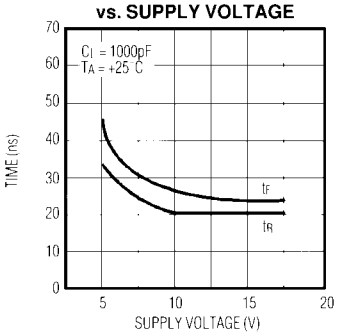
(V_{DD} = +4.5V to +18V, T_A = T_{MIN} to T_{MAX}, unless otherwise specified.)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Logic 1 Input Voltage	V_{IH}			2.4			V
Logic 0 Input Voltage	V_{IL}					0.8	V
Input Current	I_{IN}	V_{IN} = 0V to 18V		-1		1	µA
Output High Voltage	V_{OH}	No load		V_{DD}-25			mV
Output Low Voltage	V_{OL}	No load				25	mV
Output Resistance	R_{OUT}	V_{DD} = 18V, I_{LOAD} = 10mA	V_{IN} = 0.8V for inverting stages, V_{IN} = 2.4V for noninverting stages; T_A = +25°C		4	10	Ω
			T_A = T_{MIN} to T_{MAX}		5	12	
			V_{IN} = 2.4V for inverting stages, V_{IN} = 0.8V for noninverting stages; T_A = +25°C		4	10	
			T_A = T_{MIN} to T_{MAX}		5	12	
Peak Output Current	I_{PK}	V_{DD} = 18V			1.5		A
Power-Supply Current	I_{SUPP}	V_{IN} = +3V both inputs	T_A = +25°C		1.8	4.5	mA
			T_A = T_{MIN} to T_{MAX}		2.5	8.0	
		V_{IN} = 0V both inputs	T_A = +25°C		0.2	0.4	
			T_A = T_{MIN} to T_{MAX}		0.3	0.6	
Rise Time (Note 1)	t_R	T_A = +25°C			20	30	ns
		T_A = T_{MIN} to T_{MAX}			25	40	
Fall Time (Note 1)	t_F	T_A = +25°C			20	30	ns
		T_A = T_{MIN} to T_{MAX}			25	40	
Delay Time (Note 1)	t_{D1}	T_A = +25°C			10	30	ns
		T_A = T_{MIN} to T_{MAX}			15	40	
	t_{D2}	T_A = +25°C			25	50	
		T_A = T_{MIN} to T_{MAX}			30	60	

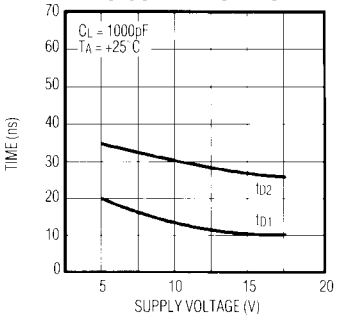
Note 1: Switching times guaranteed by design, not tested. See Figure 1 for timing measurement circuit.

Typical Operating Characteristics

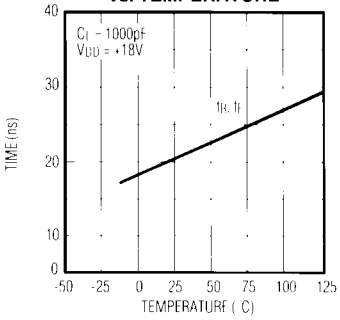
MAX4426
RISE AND FALL TIME
vs. SUPPLY VOLTAGE



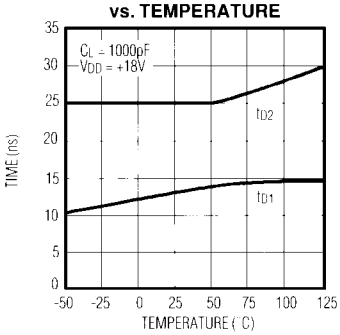
MAX4426
DELAY TIME
vs. SUPPLY VOLTAGE



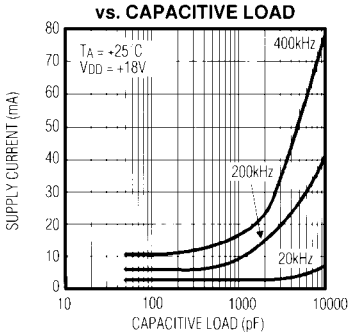
MAX4426
RISE AND FALL TIME
vs. TEMPERATURE



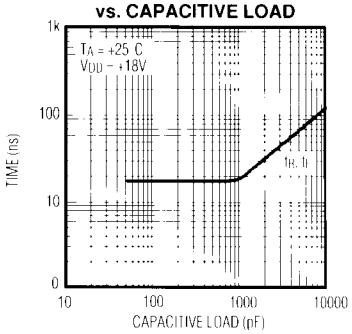
MAX4426
DELAY TIME
vs. TEMPERATURE



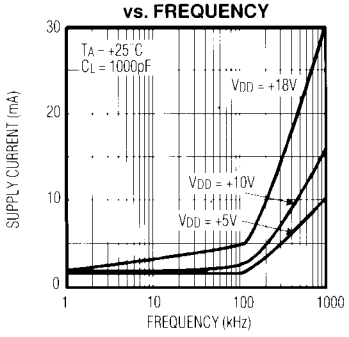
MAX4426
SUPPLY CURRENT
vs. CAPACITIVE LOAD



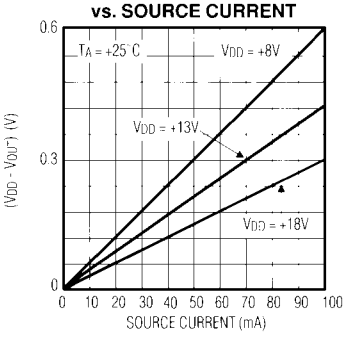
MAX4426
RISE AND FALL TIME
vs. CAPACITIVE LOAD



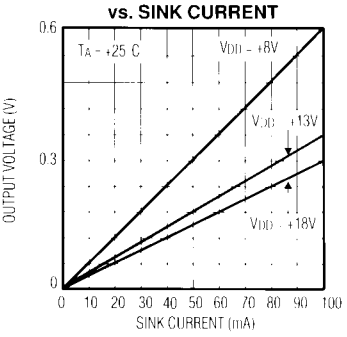
MAX4426
SUPPLY CURRENT
vs. FREQUENCY



MAX4426
OUTPUT HIGH VOLTAGE
vs. SOURCE CURRENT



MAX4426
OUTPUT LOW VOLTAGE
vs. SINK CURRENT



Dual High-Speed 1.5A MOSFET Drivers

Applications Information

The MAX4426/4427/4428 have easy-to-drive inputs. However, these inputs must never be allowed to stay between V_{IH} and V_{IL} for more than 50ns. Unused inputs should always be connected to ground to minimize supply current. Drivers can be paralleled on the MAX4426 or MAX4427 by tying both inputs together and both outputs together.

Supply bypassing and grounding are extremely important with the MAX4426/4427/4428, as the peak supply current can be as high as 3A, which is twice the peak output current. Ground drops are a form of negative feedback with inverters, and hence will degrade the delay and transition time of the MAX4426/MAX4428.

Suggested bypass capacitors are a 4.7µF (low ESR) capacitor in parallel with a 0.1µF ceramic capacitor, mounted as close as possible to the MAX4426/4427/4428. Use a ground plane if possible or separate ground returns for inputs and outputs. Output voltage ringing can be minimized with a 5Ω to 20Ω resistor in series with the output, but this will degrade output transition time. Ringing may be undesirable due to the large current that flows through capacitive loads when the voltage across these loads transitions quickly.

Operation at the upper end of the supply voltage range (>15V) requires that a capacitance of at least 50pF be present at the outputs. This prevents the supply voltage provided to the die (which can be different from that seen at the IC supply pin) from exceeding the 20V absolute maximum rating, due to overshoot. Since at least 50pF of gate capacitance is present in all higher power FETs, this requirement is easily met.

Power Dissipation

The MAX4426/4427/4428 power dissipation consists of input inverter losses, crowbar current through the output devices, and output current (either capacitive or resistive). The sum of these must be kept below the maximum power dissipation limit.

The DC input inverter supply current is 0.2mA when both inputs are low and 2mA when both inputs are high. The crowbar current through an output device making a transition is approximately 100mA for a few nanoseconds. This is a small portion of the total supply current, except for high switching frequencies or a small load capacitance (100pF).

The MAX4426/4427/4428 power dissipation when driving a ground-referenced resistive load is:

$$P = (D)\ (r_{ON(MAX)})\ (I_{LOAD}^2)$$

where D is the percentage of time the MAX4426/4427/4428 output pulls high, $r_{ON(MAX)}$ is the MAX4426/4427/4428 maximum on resistance, and I_{LOAD} is the MAX4426/4427/4428 load current.

For capacitive loads, the power dissipation is:

$$P = (C_{LOAD})\ (V_{DD}^2)\ (FREQ)$$

where C_{LOAD} is the capacitive load, V_{DD} is the MAX4426/4427/4428 supply voltage, and FREQ is the toggle frequency.

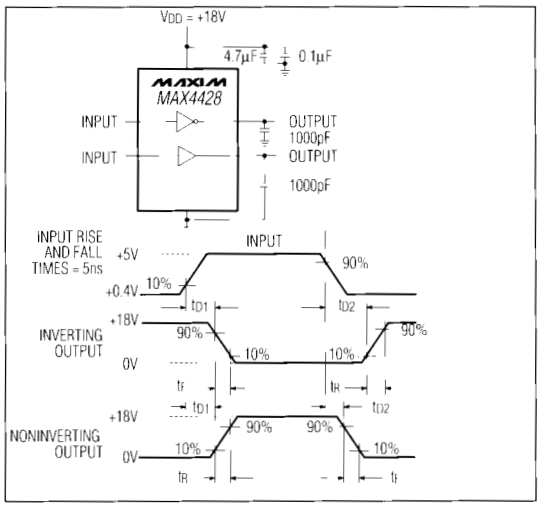


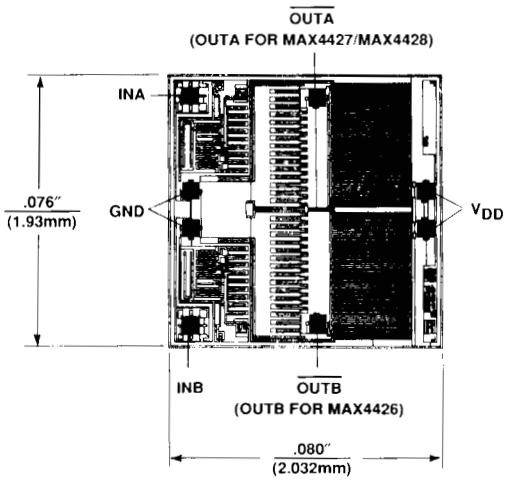
Figure 1. Inverting and Noninverting Test Circuit

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX4427CPA	0°C to +70°C	8 Plastic DIP
MAX4427CSA	0°C to +70°C	8 SO
MAX4427C/D	0°C to +70°C	Dice*
MAX4427EPA	-40°C to +85°C	8 Plastic DIP
MAX4427ESA	-40°C to +85°C	8 SO
MAX4427EJA	-40°C to +85°C	8 CERDIP
MAX4427MJA	-55°C to +125°C	8 CERDIP**
MAX4428CPA	0°C to +70°C	8 Plastic DIP
MAX4428CSA	0°C to +70°C	8 SO
MAX4428C/D	0°C to +70°C	Dice*
MAX4428EPA	-40°C to +85°C	8 Plastic DIP
MAX4428ESA	-40°C to +85°C	8 SO
MAX4428EJA	-40°C to +85°C	8 CERDIP
MAX4428MJA	-55°C to +125°C	8 CERDIP**

** Dice are tested at T_A = +25°C.*
*** Contact factory for availability and processing to MIL-STD-883.*

Chip Topography

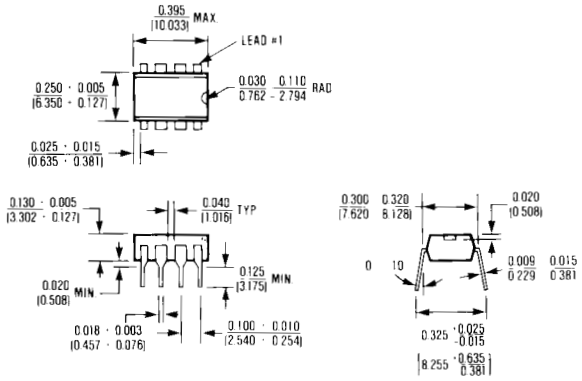


SUBSTRATE CONNECTED TO V_{DD};
TRANSISTOR COUNT: 26.

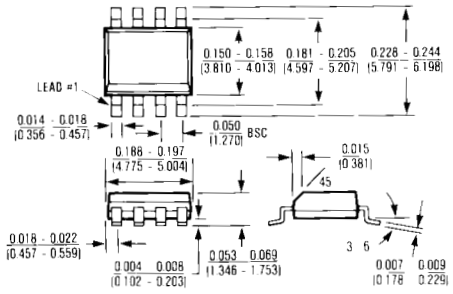
MAX4427/MAX4428

Dual High-Speed 1.5A MOSFET Drivers

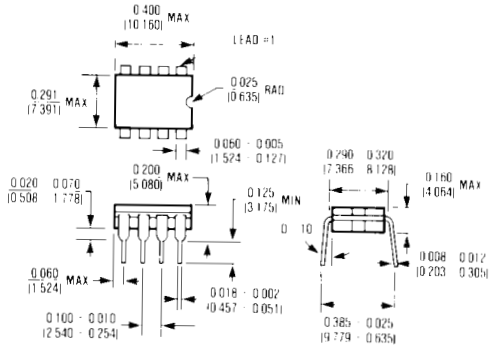
Package Information



8 Lead Plastic DIP
θ_{JA} = 120° C/W
θ_{JC} = 70° C/W



8 Lead Small Outline
θ_{JA} = 170 C/W
θ_{JC} = 80 C/W



8 Lead CERDIP
θ_{JA} = 125 C/W
θ_{JC} = 55 C/W

DIOTEC ELECTRONICS CORP.
18020 Hobart Blvd., Unit B
Gardena, CA 90248 U.S.A
Tel.: (310) 767-1052 Fax: (310) 767-7958

Data Sheet No. GPDP-601-1C

6 AMP GENERAL PURPOSE SILICON DIODES

FEATURES

- Low cost
- Low leakage
- Low forward voltage drop
- High current capacity
- Easily cleaned with freon, alcohol, chlorothene and similar solvents

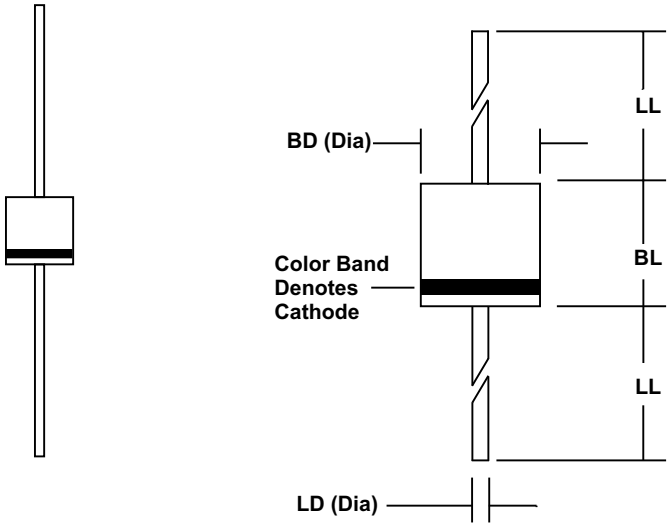
MECHANICAL DATA

- Case: Molded epoxy (U/L Flammability Rating 94V-0)
- Terminals: Plated axial leads
- Soldering: Per MIL-STD 202 Method 208 guaranteed
- Polarity: Color band denotes cathode
- Mounting Position: Any
- Weight: 0.07 Ounces (2.1 Grams)

MECHANICAL SPECIFICATION

ACTUAL SIZE OF GP600 PACKAGE

SERIES GP600 - GP610



Sym	Minimum		Maximum	
	In	mm	In	mm
BL	0.340	8.6	0.360	9.1
BD	0.340	8.6	0.360	9.1
LL	1.00	25.4		
LD	0.048	1.2	0.052	1.3

MAXIMUM RATINGS & ELECTRICAL CHARACTERISTICS

Ratings at 25 °C ambient temperature unless otherwise specified.
Single phase, half wave, 60Hz, resistive or inductive load.
For capacitive loads, derate current by 20%.

PARAMETER (TEST CONDITIONS)	SYMBOL	RATINGS							UNITS
Series Number		GP600	GP601	GP602	GP604	GP606	GP608	GP610	
Maximum DC Blocking Voltage	V_{RM}	50	100	200	400	600	800	1000	VOLTS
Maximum RMS Voltage	V_{RMS}	35	70	140	280	420	560	700	
Maximum Peak Recurrent Reverse Voltage	V_{RRM}	50	100	200	400	600	800	1000	
Average Forward Rectified Current @ T_A = 60 °C, Lead length = 0.375 in. (9.5 mm)	I_O	6							AMPS
Peak Forward Surge Current (8.3 mSec single half sine wave superimposed on rated load)	I_{FSM}	400							
Maximum Forward Voltage at 6 Amps DC	V_{FM}	1							VOLTS
Maximum Full Cycle Reverse Current @ T_L = 75 °C (Note 1)	$I_{RM(AV)}$	25							µA
Maximum Average DC Reverse Current @ T_A = 25°C At Rated DC Blocking Voltage @ T_A = 100°C	I_{RM}	10 100							
Typical Thermal Resistance, Junction to Ambient (Note 1)	$R_{\theta JA}$	10							°C/W
Typical Junction Capacitance (Note 2)	C_J	100							pF
Operating and Storage Temperature Range	T_J, T_{STG}	-65 to +175							°C

01.00 fgrpgp601

NOTES: (1) Lead length = 0.375 in. (9.5 mm)
(2) Measured at 1MHz & applied reverse voltage of 4 volts

DIOTEC ELECTRONICS CORP.
18020 Hobart Blvd., Unit B
Gardena, CA 90248 U.S.A
Tel.: (310) 767-1052 Fax: (310) 767-7958

6 AMP GENERAL PURPOSE SILICON DIODES

RATING & CHARACTERISTIC CURVES FOR SERIES GP600 - GP610

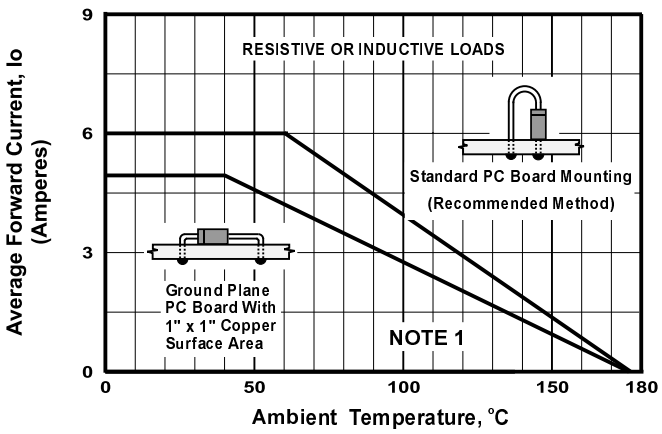


FIGURE 1. FORWARD CURRENT DERATING CURVE

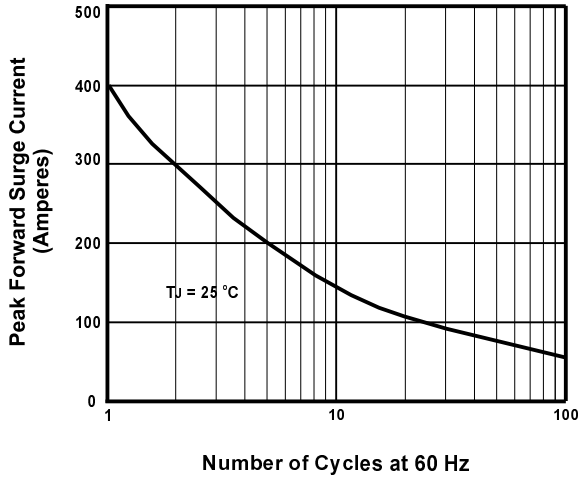


FIGURE 2. MAXIMUM NON-REPETITIVE SURGE CURRENT

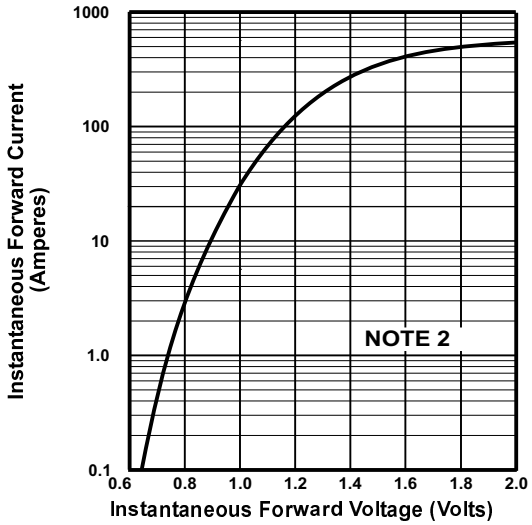


FIGURE 3. TYPICAL FORWARD CHARACTERISTICS

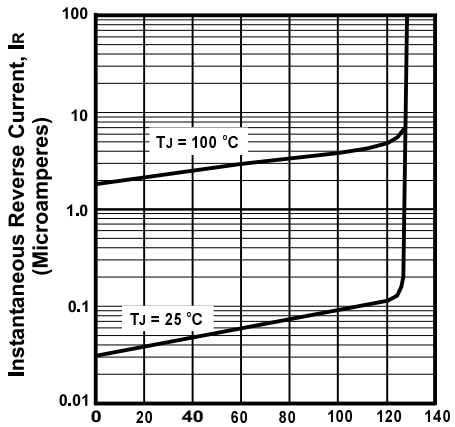


FIGURE 4. TYPICAL REVERSE CHARACTERISTICS

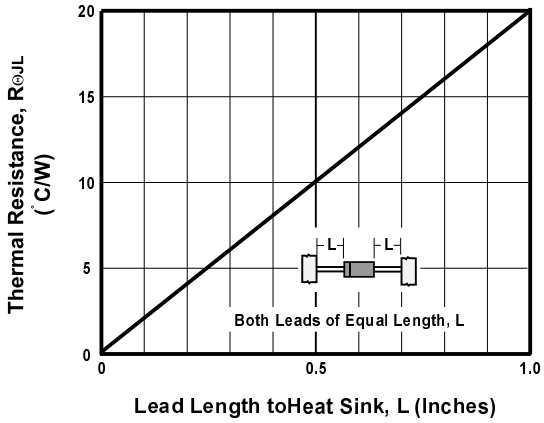


FIGURE 5. TYPICAL THERMAL RESISTANCE

NOTES

(1) Single Phase, Half Wave, 60 Hz

(2) TJ = 25 °C, Pulse Width = 300 μSec, 1.0% Duty Cycle

4.97 bg pdp 601