Guía Didáctica del 68HC08

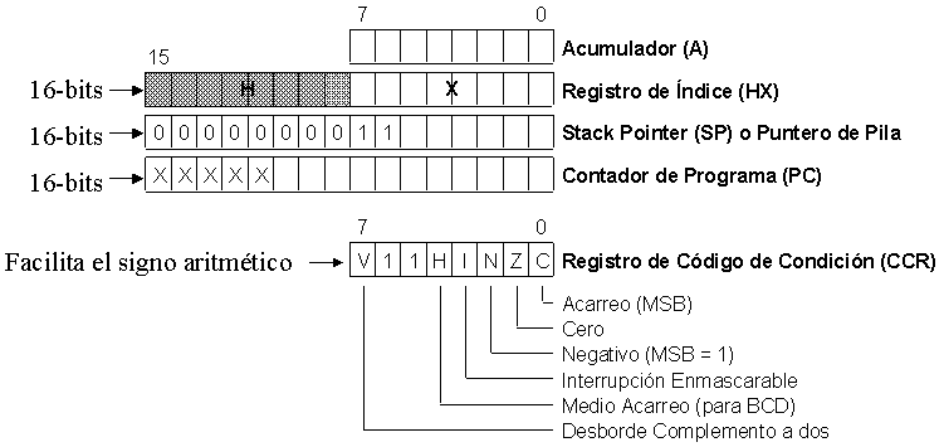
Jordi Mayné
Ingeniero de Aplicaciones

Rev. 1

Índice

Arquitectura de la CPU 68HC08

El modelo de programación de la CPU del 68HC08 es una evolución del modelo de CPU del 68HC05. Cada 68HC08 implementa este modelo de CPU sin tener en cuenta el tamaño o el conjunto de características propias. A continuación se verán las mejoras específicas que se incluyen en la arquitectura 68HC08:



- El *registro de índice* se ha ampliado a 16-bits, ayudando a manejar mejor las tablas o estructuras de datos que son mayores de 256 bytes.
- El *puntero de pila* y el *contador de programa* son registros de 16-bits, sin tener en cuenta la memoria interna disponible. Más adelante se verá la pila ('stack').
- En *Power-On Reset*, el 68HC08 se parece al 68HC05. Durante el Reset, los 8 bits más altos del registro de índice HX, de los 16 bits que tiene, se ponen a cero y el puntero de pila se inicializa a $00FF como en el 68HC05. Considerando que la mayoría de 68HC08 tienen mayor cantidad de RAM disponible, es probable que el usuario relocalizará el puntero de pila ('stack pointer'). Sin embargo, este característica ayuda a mantener la compatibilidad operacional con el software existente del 68HC05.
- En la CPU del 68HC08 el *bit V,* del registro de código de condición (CCR) facilita los cálculos aritméticos con signo. Esta mejora permite a los programadores de lenguaje ensamblador y a los compiladores, realizar cálculos de direccionamiento mucho mejor.

Compatibilidad de Código 68HC05/08

El código objeto ('opcode') del 68HC08 es compatible con el del 68HC05. Usa los mismos modelos de 'opcodes' binarios. No se necesita modificar el código fuente del 68HC05 para ponerlo en el 68HC08. Se puede descargar el mismo fichero S-Record existente del 68HC05 y se puede esperar que el software se ejecute correctamente en el 68HC08.

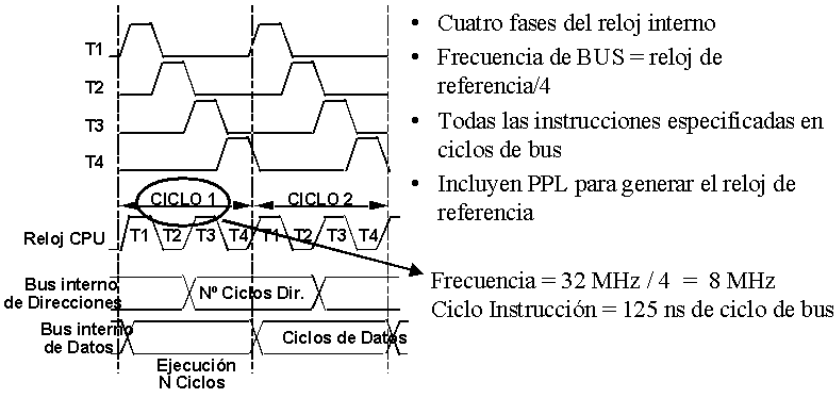
Hay dos diferencias importantes entre el 68HC05 y el 68HC08 que se deben considerar al poner a punto el código. Primero, hay alguna diferencia en el direccionamiento y en los periféricos, pero esto afecta a una cantidad porcentual pequeña de código. Los estados de máquina, las matemáticas y los algoritmos de control de flujo, son transparentes en código. Segundo, el 68HC08 ejecutará el código del 68HC05 aproximadamente el 25% más rápido, corriendo a la misma velocidad de bus, y el 68HC08 pueden correr típicamente cuatro veces más rápido que el 68HC05. Esto hace un promedio de 5 veces de actuación más rápida que el código ejecutado en el 68HC05. Puesto que el 68HC08 es mucho más rápido que el 68HC05, los más probable, es que se necesite actualizar el tiempo de los bucles implementados en el software del 68HC05.

El 68HC08 incluye un conjunto de instrucciones y modos de direccionamiento mejorados, que pueden proporcionar mayores prestaciones. Después se verán estas características y cómo se pueden utilizar para aumentar las prestaciones con un tamaño de código más pequeño.

Tiempos de los Ciclos Internos de la CPU 68HC08

El 68HC08 utiliza cuatro fases del reloj interno en cada ciclo de ejecución de la CPU. Si el 68HC08 está gobernado por un cristal, el ciclo de ejecución es un cuarto de la frecuencia del cristal. A este ciclo se le llama *ciclo de bus* o *ciclo de instrucción*.

En el 68HC08, todos los tiempos de cada instrucción se especifican en *ciclos de bus*. Por ejemplo, un reloj de entrada de 32 MHz producirá una frecuencia de bus de 8 MHz. Un *ciclo de bus* de una instrucción se ejecutará en 125 ns o 1 dividido por 8 MHz.

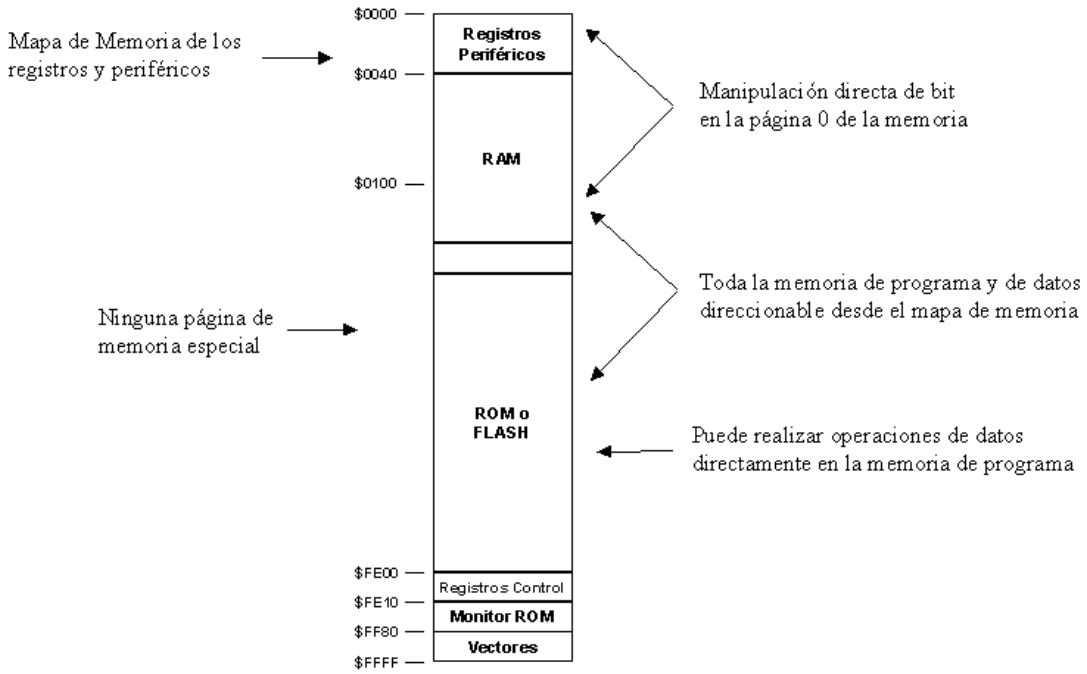


La mayoría de 68HC08 incluyen un circuito PLL interno que a partir de una frecuencia más baja de cristal, genera internamente una frecuencia de reloj mucho más alta que la del cristal. Para más información sobre la generación del reloj y el circuito PLL, se puede ver en el Módulo de Generación de Reloj (CGM) de esta guía didáctica.

Organización de la Memoria del 68HC08

La figura siguiente muestra el mapa de memoria del 68HC08. Este mapa de memoria es idéntico al empleado para el 68HC05, sólo que se ha implementado hasta 64 kbytes, sin tener en cuenta los diferentes tamaños de memoria de cada 68HC08 disponible.

En el 68HC08, el mapa de memoria empieza con los registros de los periféricos con 64 bytes. A continuación le sigue el espacio para la RAM. La ROM o FLASH ocupa la parte superior de la memoria que precede a $FE00, donde están los Vectores, el programa Monitor y los Registros de control. En el medio del mapa de memoria, entre la RAM y la ROM/FLASH, hay una zona de memoria no utilizable por el usuario que realiza las verificaciones de direcciones ilegales.



Todas las MCU de la familia 68HC08 usan este modelo, aunque algunas de ellas tienen una dirección de inicio y final para las áreas específicas del monitor ROM o los registros de control.

Juego de Instrucciones del 68HC08

El 68HC05 tiene un total de 85 instrucciones que en la actualidad todavía forman un juego básico de instrucciones funcional y potente. El juego de instrucciones del 68HC08 se ha construido sobre la base de las instrucciones del 68HC05. El 68HC08 añade 28 instrucciones al juego de instrucciones del 68HC05, remarcadas en la tabla siguiente. Muchas de estas instrucciones soportan el registro del índice extendido a 16 bits y el puntero de pila (‘stack pointer’) es totalmente relocalizable.

Movimiento de Datos	LDA, LDX, STA, STX, TAX, TXA, **LDHX, MOV, PSHA, PSHH, PSHX, PULA, PULH, PULX, STHX**
Aritméticas	ADD, ADC, SUB, SUBC, MUL, **DAA, DIV**
Manipulación de Datos (Inc/Dec/Neg/Clr)	INCA, INCX, INC, DECA, DECX, DEC, CLR,NEGA, NEGX, NEG, **AIS, AIX, CLRH**
Manipulación de Datos (Rotación/Desplazamiento)	ROLA, ROLX, ROL, RORA, RORX, ROR, LSLA, LSLX, LSL,LSRA, LSRX, LSR, ASRA, ASRX, ASR
Manipulación de Datos	BSET, BCLR
Lógicas	AND, ORA, EOR, COMA, COMX, COM, **NSA**
Test de Datos	CMP, CPX, BIT, TSTA,TSTX, TST, BRCLR, BRSET, **CPHX**
Bifurcación	BRA, BRN, BSR, BHI, BLO, BHS, BLS, BPL, BMI, BEQ, BNE, BCC, BCS, BHC, BHCC, BHCS, BMC, BMS, BIL, BIH, **BGE, BGT, BLE, BLT, CBEQ, CBEQA, CBEQX, DBNZ**
Salto/Retorno	JMP, JSR, RTS
Control	SEC, CLC, SEI, CLI, SWI, RTI, RSP, NOP, WAIT, STOP, **TAP, TPA, TSX, TXS**

Se han añadido varias instrucciones de bifurcación condicional, como: el ajuste decimal del acumulador DAA y la instrucción NSA (Nibble Swap Accumulator). La instrucción de división DIV sin signo, dividiendo 16 por 8 ejecutándose en sólo 7 ciclos de bus. La instrucción multiplicación (MUL) se ha reforzado para que se ejecute en sólo 5 ciclos de bus, en lugar de los once ciclos necesarios para el 68HC05. La instrucción CLRA para Borrar el Acumulador es de un sólo ciclo para el 68HC08, mientras que el 68HC05 necesita 3 ciclos.

Esta mejora del juego de instrucciones del 68HC08, proporciona mayor eficacia de código y funcionamiento, recogiendo múltiples instrucciones y realizando la misma tarea con una sola instrucción. Más adelante se verán en detalle estas nuevas instrucciones.

Modos de Direccionamiento del 68HC05

Para comprender mejor los modos de direccionamiento del 68HC08, es mejor empezar viendo algunos ejemplos de los modos de direccionamiento del 68HC05, como el modo de direccionamiento inherente, inmediato, directo, extendido, relativo e indexado. Estos modos de direccionamiento tanto se pueden usar en el 68HC05 como en el 68HC08.

Inherente: CLRA

Las instrucciones de ***direccionamiento inherente*** no tienen ningún operando, ya que el operando se define en el 'opcode' de 8-bits. Por ejemplo para borrar el acumulador, se usa la instrucción CLRA, que es una instrucción de un sólo ciclo para el HC08; el HC05 toma 3 ciclos.

Inmediato: LDA #20

Las instrucciones de ***direccionamiento inmediato*** tienen los operandos, que siguen inmediatamente al 'opcode', de 8-bits o de 16-bits. Cargar el acumulador con 20 es una instrucción de dos bytes, que se ejecuta en 2 ciclos de bus.

Directo: LDA $40

Las instrucciones de ***direccionamiento directo*** tienen la dirección de 8-bits del operando que sigue inmediatamente al 'opcode'. Por consiguiente, las instrucciones acceden directamente a los primeros 256 bytes de la memoria. Anteriormente, se hizo referencia a este tipo de acceso como página directa o página cero. Cargar el acumulador con el operando a la posición de memoria 40, es una instrucción de dos bytes que se ejecuta en 3 ciclos de bus.

Extendido LDA $4000

Las instrucciones de ***direccionamiento extendido*** proporcionan direccionamiento absoluto a cualquier posición en los 64K del mapa de memoria, sin paginar. Requieren en total tres bytes para el 'opcode' más la dirección de 16-bits del operando. La mayoría de los ensambladores usan el modo directo más corto automáticamente, para cualquier acceso a los primeros 256 bytes del mapa de memoria.

Relativo (± 128) BLT LOOP

Todas las instrucciones de bifurcación condicional usan el ***direccionamiento relativo***. Si la condición de bifurcación es verdad, el contador de programa se agrega al byte con signo, que sigue inmediatamente al 'opcode' de bifurcación. Esto da un rango de -128 a +127 bytes, para la bifurcación. Las instrucciones de salto o de salto a subrutina, se pueden usar para mover el contador de programa en cualquier parte de los 64K del mapa de memoria.

Indexado Sin 'Offset' LDA ,X
Indexado Con 'Offset' de 8 bits LDA $40, X
Indexado Con 'Offset' de 16 bits LDA $4000, X

Los modos de ***direccionamiento indexado*** son claves para direccionar tablas y otras estructuras de datos de una manera eficaz. El direccionamiento indexado sin desplazamiento ('offset'), se refiere a lo que en la mayoría de otras arquitecturas es el direccionamiento del puntero indirecto. El valor en el registro de índice es la dirección o el puntero del operando. Los modos de direccionamiento con 'offset' proporcionan al 68HC08 una gran eficacia de código comparado con otras arquitecturas con sólo direccionamiento indirecto o direccionamiento indirecto con otro registro de 'offset'.

Ejemplo de Direccionamiento Indexado con offset de 8 bits

A + B = C, donde cada letra es una tabla de 5 valores. Se tienen dos tablas de datos y se añade a cada entrada de la tabla **A**, su entrada correspondiente en la tabla **B** y entonces se guarda el resultado en la tabla **C**.

	LDX	#5	2 ciclos
LOOP	LDA	A_Data - 1, X	3 ciclos
	ADD	B_Data - 1, X	3 ciclos
	STA	C_Result - 1, X	3 ciclos
	DBNZX	LOOP	3 ciclos
			---
			2 ciclos
			+ 12 ciclos del bucle

Total: 10 bytes de código

A_Data
71
02
20
FC
9D

B_Data
08
A3
7B
01
3C

C_Result
xx
xx
xx
xx
xx

Este primer ejemplo es muy básico, pero muestra la potencia del modo de direccionamiento indexado con un 'offset' de 8-bits. La primera instrucción carga el registro de índice X, con 5. Éste será el 'offset' en cada entrada de las tablas, así como el contador para hacer un bucle para cada entrada. Seguidamente, la instrucción LDA carga el acumulador con la quinta entrada de la tabla A, 9D. El indicador de dirección para esta instrucción se ha calculado primero para el ensamblador, restando 1 a la etiqueta de dirección A_Data. Entonces la CPU añade el valor 5 al registro de índice para apuntar a la quinta entrada de la tabla. La instrucción ADD añade 9D en el acumulador con la quinta entrada de la tabla B, 3C. El resultado se pone en el acumulador. La instrucción STA guarda el resultado a la quinta entrada de la tabla C. El direccionamiento indexado con 'offset', permite con un simple puntero direccionar tres tablas sin consumir tiempo ni tamaño de código, en recargar el puntero.

La instrucción DBNZX es una nueva instrucción del 68HC08 que decrementa el registro X y bifurca si el resultado no es igual a cero. Esto decrementará el puntero a 4 y bifurcará hacia atrás a la instrucción LOOP, para realizar la misma operación en la cuarta entrada de cada tabla. Esto continúa hasta que todas las cinco entradas se completan con X = 0. Más tarde, se verán algunas nuevas instrucciones de direccionamiento del 68HC08 que activa el decremento del acumulador o un operando, en cualquier parte de los 64K del mapa de memoria. Esta rutina utiliza 2 ciclos más 12 ciclos por bucle y un total de tan sólo 10 bytes de espacio de código. Esto es mucho mejor que cargar y recargar el puntero, manteniendo una cuenta separada del bucle y verificando la cuenta de cada bucle.

Ejemplo de Direccionamiento Indexado con offset de 16 bits

A + B = C, donde cada letra es una tabla de 5 valores. Las tablas pueden estar en cualquier parte de la memoria, dentro de los 64K del mapa de memoria.

	LDX	#5	2 ciclos
LOOP	LDA	A_Data - 1, X	4 ciclos
	ADD	B_Data - 1, X	4 ciclos
	STA	C_Result - 1, X	4 ciclos
DBNZX	LOOP		3 ciclos
			---
			2 ciclos
			+ 15 ciclos por bucle

Total: 13 bytes de código

A_Data
71
02
20
FC
9D

B_Data
08
A3
7B
01
3C

C_Result
Xx
Xx
Xx
Xx
Xx

En el ejemplo anterior se asumió que las tablas se pueden direccionar con un 'offset' de 8-bits. Ahora se realiza la misma función con tablas que se localizan en cualquier parte de la memoria. Esto no supone ningún cambio en el código ensamblador. El ensamblador usa automáticamente el modo de direccionamiento indexado con un 'offset' de 16-bits, en lugar del 'offset' de 8-bits. Para cada instrucción, se añade un byte más y un ciclo de bus de más, tal como se muestra en el ejemplo.

Este ejemplo se mantiene igual para cualquier posición de la tabla; pueden estar en cualquier parte de los 64 Kbytes del mapa de memoria, tanto si las tablas están en la RAM o en la FLASH. Comparado con muchas arquitecturas de 8 bits, esto elimina la necesidad de código adicional para manejar el paginado con instrucciones especiales, para primero acceder a los datos guardados en la tabla del espacio de código de programa o de código adicional, en el caso de cuando la tabla supera el límite de los 256 bytes.

Nuevos Modos de Direccionamiento del 68HC08

El 68HC08 tiene varios nuevos modos de direccionamiento y son: dos modos de direccionamiento indexado, dos modos de direccionamiento del puntero de pila y cuatro modos de direccionamiento de movimiento de memoria a memoria. A continuación se muestran los modos diferentes con ejemplos de sus instrucciones:

Indexado:

–Indexado sin 'Offset' con incremento posterior CBEQ X+, Label

–Indexado con 8 bits de 'Offset' con incremento posterior CBEQ $50, X+, Label

Los dos modos de direccionamiento indexado con post-incremento automático del puntero de índice.

Stack Pointer:

El nuevo modo de direccionamiento relativo de puntero de pila ('stack pointer'), mejora en eficacia el código C y hay de dos tipos: el modo relativo del 'stack pointer' con un 'offset' de 8 bits y el modo relativo del 'stack pointer'con un 'offset' de 16-bits.

– Stack Pointer con 'offset' de 8 bits STA $10, SP

– Stack Pointer con 'offset' de 16 bits STA $1000, SP

Su trabajo es similar al modo de direccionamiento indexado, pero usan el 'stack pointer' en lugar del registro de índice H:X. Se podría usar el 'stack pointer', como un registro de índice adicional cuando las interrupciones son inválidas.

En programas en ensamblador y C, el 'stack' o pila se usa para pasar normalmente los operandos a las subrutinas. Típicamente la subrutina sacará los operandos del acumulador tal como los necesita. El direccionamiento relativo del 'stack' permite el fácil acceso de datos en el 'stack', proporcionando el acceso directo a los operandos, eliminando código y el tiempo requerido para almacenar valores en la pila ('push') y recuperar valores de la pila ('pop'), a y desde el 'stack'.

Mover de memoria a memoria:

Hay cuatro nuevos modos de direccionamiento de memoria a memoria usan la nueva instrucción de movimiento (MOV) para mover datos directamente sin usar el acumulador.

–**Mover de inmediato a directo** MOV #20,$40

El modo de movimiento inmediato a directo se usa típicamente para inicializar variables y registros en la página directa. Eliminando las instrucciones requeridas para guardar el acumulador, cargar el acumulador con los datos a transferir, guardar el acumulador con el dato a transferir y además restaurar el acumulador, reduciendo el número de ciclos de ejecución de nueve a cuatro.

–**Mover de directo a directo** MOV $20, $40

El modo de movimiento directo a directo se usa típicamente para mover datos de registro a registro desde la página directa.

–**Mover de indexado a directo con incremento posterior** MOV X+, SCDR

El modo de indexado a directo con post-incremento se usa típicamente para transferir tablas de cualquier parte de los 64K del mapa de memoria a un registro de la página directa. Por ejemplo, se puede usar este modo para transferir un 'buffer' de la RAM al transmisor SCI.

–**Mover de directo a indexado con incremento posterior** MOV SCDR, X+

De un modo similar es el movimiento de directo a indexado con post-incremento. Este modo se puede usar para transmitir datos del SCI al 'buffer' receptor.

Ejemplo de Direccionamiento Indexado

Subrutina que busca 'espacios' en una cadena de caracteres ASCII. Se supone que la cadena por lo menos contiene un carácter de no-espacio.

Entrada: H:X punto de inicio de la cadena
Salida: H:X punto al primero carácter no-espacio de la cadena

```
Start   LDA    #$20      ;Cargar el carácter buscado          2 ciclos
Skip    CBEQ   X+, Skip  ;Incrementar a través de la cadena   4 ciclos
                         ;hasta encontrar un carácter no-espacio
        AIX    #-1       ;Decrementar H:X                     2 ciclos
        RTS              ;Retorno                             4 ciclos
                                                             ----------
                                                              8 ciclos
        Total: 7 bytes de código                          + 8 ciclos por bucle
```

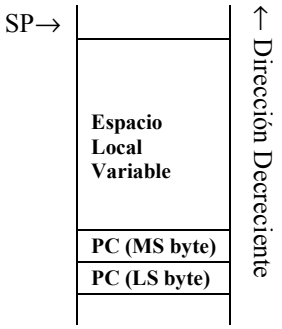
Este ejemplo se muestra el direccionamiento indexado sin 'offset' y con post-incremento. La subrutina usa la nueva instrucción del 68HC08 (CBEQ), que compara y bifurca si es igual. Este ejemplo busca una cadena de caracteres ASCII que se localizan en cualquier parte de los 64K del mapa de memoria, hasta encontrar el primer carácter que no está en blanco.

Primero, se carga el acumulador con el carácter ASCII $20, espacio en blanco. La instrucción CBEQ compara el acumulador con el operando apuntando al registro de índice H:X. Si el acumulador se iguala con el operando, indica un carácter en blanco y bifurca hacia atrás. El registro H:X se incrementa automáticamente como parte de la instrucción. Cuando bifurca hacia atrás, comparará el siguiente carácter en la cadena. Si el acumulador no se iguala con el operando, es que se ha encontrado el primer carácter de no-espacio. En este caso, el registro H:X se incrementa pero todavía la subrutina no bifurca a la instrucción de salto. En cambio, la subrutina decrementa el registro H:X con la suma inmediata a, instrucción AIX e vuelve apuntar H:X al primer carácter de no-espacio.

Este ejemplo se ejecuta con 4 ciclos de bus por carácter que se busca y ocho ciclos adicionales de cabecera. Esta subrutina representa sólo 7 bytes de código. La instrucción CBEQ también se puede usar en el modo de direccionamiento con 8 bits de 'offset' con post-incremento. Esto permite comparar operandos en cadenas múltiples de caracteres o estructuras de datos usando un simple registro de índice, tal como se vio en el ejemplo anterior. Hay también una variación de esta instrucción que compara los 8-bits del registro de índice X con operandos en memoria, llamada CBEQX.

Ejemplo de Direccionamiento del Stack Pointer

```
Sub1    AIS     # -16    ;Crea 16 bytes
                         ;de espacio local
 .
 .
Skip    AIS     #16      ;Borra el Stack Pointer
        RTS              ;Retorno
```



El 'Stack Pointer' siempre debe apuntar al próximo byte sin usar. No se usa este byte (0,SP), para el almacenamiento. El diagrama anterior muestra el direccionamiento del puntero de pila. La subrutina usa la nueva instrucción AIS de suma inmediata a la pila.

La instrucción AIS, añade inmediatamente un valor de 8-bits con signo a la pila. Se puede usar para crear y quitar un 'stack frame buffer' para guardar variables temporales. Normalmente se refiere a un asignamiento dinámico de la RAM dentro del código o de la subrutina. En lugar de que cada subrutina necesite posiciones de RAM especializadas para las variables temporales, las rutinas usan la pila sólo como caso necesario. Esto ahorra espacio de RAM en aplicaciones que tienen numerosas subrutinas que requieren almacenamiento temporal.

La instrucción AIS no pone códigos de condición, para que no afecte a ningún código de la condición pendiente. Por ejemplo, asumiendo que se tienen diez subrutinas y que cada subrutina requiere diez bytes para el almacenamiento temporal local. En lugar de asignar 100 bytes, se puede usar la pila y asignar dinámicamente la RAM. Este método usa sólo 10 bytes.

Estas técnicas permiten que se escriba código C y código modular en ensamblador, de manera muy eficaz. El puntero de pila siempre debe apuntar al próximo byte sin usar. Por consiguiente, no se usa este byte, SP compensado por 0, para el almacenamiento.

Ejemplo de Mover de Memoria a Memoria

```
                                    ; Mover de indexado a directo
SIZE            EQU     16          ; Tamaño del buffer circular TX
SCSR1           EQU     $16         ; Registro de estado del SCI
SCDR            EQU     $18         ; Registro de transmisión de datos del SCI
                ORG     $50
PTR_OUT         RMB     2           ; Buffer circular de datos fuera del puntero
TX_B            RMB     SIZE        ; Buffer circular
                ORG     TX_IRQ      ; Transmisión desocupada del servicio de interrupción
                                    ; del SCI
TX_INT          LDHX    PTR_OUT     ; Carga el puntero
                LDA     SCSR1       ; Lectura inicial del registro de estado SCTE del SCI
                MOV     X+, SCDR    ; Mueve el nuevo byte al SCI e incrementa H:X
                CPHX    #TX_B + SIZE ; ¿Ha ido más allá del final del buffer circular?
BLS             NOLOOP              ; si es así, continua
                LDHX    #TX_B       ; Restablece todo lo demás para empezar de nuevo
                                    ; en el buffer
NOLOOP  STHX    PTR_OUT             ; Guarda el valor del nuevo puntero
                RTI                 ; Retorno
```

Este ejemplo es una rutina que maneja una interrupción asíncrona del puerto de transmisión serie del SCI que tiene un 'buffer' circular. La rutina utiliza el indicador de interrupción de transmisión vacía del SCI. En el Módulo del SCI se puede ver en detalle. Se puede examinar el código que la CPU necesita, para realizar la rutina de servicio de interrupción. La rutina muestra el modo de direccionamiento de mover de indexado a directo y el modo de direccionamiento de memoria a memoria.

El ejemplo usa un 'buffer' serie de transmisión de 16 bytes. Se han asignado 2 bytes en la RAM para el puntero del 'buffer' transmisor y se han asignado 16 bytes para el propio 'buffer'. Cuando el SCI está listo para transmitir un byte, se genera una interrupción si se han activado las interrupciones. Cuando esto ocurre, la rutina de servicio de interrupción, carga primero el puntero en el 'buffer' circular de transmisión desde la RAM previamente asignada. A esto le sigue una lectura del registro de estado del SCI, que es el primer paso para borrar el indicador de transmisor vacío. El indicador de transmisor vacío finalmente se borra, después de la instrucción MOV que transfiere el byte actual del 'buffer' de transmisión al registro de transmisión de datos del SCI. Al mismo tiempo, el puntero H:X se incrementa automáticamente para apuntar a la siguiente posición en el 'buffer' transmisor.

Después de la instrucción MOV, la instrucción CPHX verifica si el incremento está más allá del extremo del buffer. Si no, la rutina guarda el indicador H:X actualizado y vuelve. Si el indicador se ha ido más allá del extremo del 'buffer', el indicador se restablece al principio, TX_B, para formar un buffer circular.

Modos de Bajo Consumo del 68HC08 (Wait y Stop)

Modo Wait:

Se entra ejecutando la instrucción WAIT (Consumo típico 50% de IDD en modo RUN)

Efectos:

• El reloj de la CPU se desactiva.
• El reloj del bus sigue activo.
• Los periféricos individuales se pueden desactivar para mejorar el consumo.

Se sale del modo 'Wait' por un Reset o por una interrupción externa o interna.

Modo Stop:

Se entra ejecutando la instrucción STOP (Consumo típico: I_{DD} de 1µA a 3 µA)

Efectos:

• El reloj de la CPU se desactiva.
• El reloj del Bus opcionalmente se desactiva.

Se sale del modo 'Stop' por un Reset, por una interrupción externa o por una interrupción TBM (si el reloj del bus está activado).

Ejercicio de una Rutina para Mover un Bloque

Se trata de escribir una rutina de mover un bloque para copiar datos. En las instrucciones de inicialización hay que poner el origen de los datos, con inicio en la dirección $0140. Hay que copiar los datos a la memoria empezando en la dirección $0050. El origen del programa en la dirección $6E00. En el lazo del programa principal, hay que copiar el dato de un byte en un momento determinado, incluyendo un chequeo del valor 0 del byte.

```
                ORG $140        ; Origen de los datos en la dirección $140
SOURCE          FCC 'Hola'      ; Tabla de datos que se va a mover
                FCB 0           ; Byte Constante con valor 0
DEST            EQU $50         ; Dirección de inicio del destino $50
                ORG $6E00       ; Origen del programa en la dirección $6E00
                CLRX            ; 1. Borra el registro X.
LOOP            LDA SOURCE,X    ; 2. Saca el byte de datos de la dirección fuente.
                STA DEST,X      ; 3. Escribe el byte de dato a la dirección destino.
                BEQ DONE        ; 4. Si el byte de datos movido es cero ir a 7, el resto ir a 5.
                INCX            ; 5. Incrementa el puntero X a la siguiente posición de byte.
                BRA LOOP        ; 6. Ir al paso 2.
DONE            BRA DONE        ; 7. Hecho.
```

Ejercicio para Encontrar el valor Máximo en una lista de diez bytes.

Se trata de escribir una rutina para encontrar el valor máximo en una lista de diez bytes de datos. En las instrucciones de inicialización, hay que poner el origen de los datos empezando en la dirección $130. Usar una tabla para guardar los diez bytes de datos y poner el origen del programa en la dirección $6E00. En el lazo del programa principal, hay que usar los punteros para buscar el valor máximo, incluyendo un chequeo para el valor 0 del byte.

```
        ORG     $130            ; Origen de los datos en la dirección $130
DATA    FCB     $C0,$40,$8B     ; Tabla de bytes de datos
        FCB     $75,$A0,$60
        FCB     $DB,$25,$B0
        FCB     $50

        ORG     $6E00           ; Origen del programa en la dirección $6E00
        LDX     #10             ; 1. Carga el número de bytes en el registro X.
GETBYTE LDA     DATA-1,X        ; 2. Carga el acumulador con el byte desde la dirección fuente.
DECCNT  DECX                    ; 3. Decrementa el registro X.
        BEQ     DONE            ; 4. Si el puntero = 0, entonces ir al paso 8 (Hecho)
        CMP     DATA-1,X        ; 5. Compara el valor del acumulador del byte
                                ; apuntado con el puntero fuente.
       BHS     DECCNT  ; 6. Si el valor del acumulador es mayor del valor del byte apuntado
                                ; con el puntero fuente ir al paso 3, el resto ir al paso 7.
        BRA     GETBYTE         ; 7. Lazo hacia atrás con la instrucción GETBYTE
DONE    BRA     DONE            ; 8. Hecho.
```

Ejercicio de una Rutina de Borrado de la RAM

Se trata de escribir una rutina para inicializar la RAM desde $0050 a $044F. En las instrucciones de inicialización, hay que poner el origen del programa en la dirección $6E00. Inicializar el puntero para que inicie en la posición de la memoria RAM $0050.

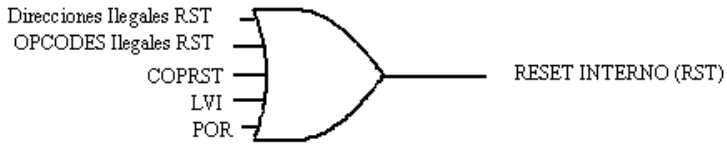
```
        ORG     $6E00           ; Origen del programa en la dirección $6E00
        LDHX    #$0050          ; 1. Inicializa el puntero a la primera posición a borrar

LOOP    CLR     ,X              ; 2. Borra la posición de memoria apuntada por el puntero.
        AIX     1               ; 3. Incrementa el puntero a la posición siguiente.
        CPHX    #$0500          ; 4. Compara el puntero con  $0500.
        BNE     LOOP            ; 5. Si el puntero  = $0500 ir al paso 6, el resto ir a paso 2.
DONE    BRA     DONE            ; 6. Hecho (bifurcar sobre si mismo).
```

Módulo del Reset y las Interrupciones

En esta parte se explica el proceso del Reset y de las Interrupciones en la MCU. Se describen las diferencias entre las Interrupciones y el Reset. Se identifican las diferentes fuentes de Interrupción y Reset. Se describe el proceso de restablecimiento después de un Reset. El Reset y las interrupciones son respuestas a los eventos excepcionales durante la ejecución de un programa.

Fuentes de Reset

- **Externas:** *Power On Reset* (POR) o con el *Pin de Reset puesto a masa*.
- **Internas:** *COP* o *LVI* o *Opcodes Ilegales* o *Direcciones Ilegales*



El Reset puede ser causado por una señal en el pin de reset externo o por una señal de reset interna. Las señales de reset internas se pueden generar por el módulo COP o por el módulo LVI. Otras fuentes de reset internas son por un 'opcode' ilegal y por una dirección ilegal.

Un reset detiene la ejecución del programa en la MCU. Una vez se ha procesado el reset, la MCU vuelve inmediatamente a las condiciones de inicio conocidos y empieza la ejecución del programa desde una posición de memoria definida por el usuario.

Proceso de un Reset

• Se genera un reset por una *dirección ilegal* cuando la CPU intenta sacar una instrucción de una dirección que no está definida en el mapa de memoria. Este tipo de reset proporciona una protección adicional del sistema y devuelve a la CPU a un estado conocido cuando se usa una dirección inválida.

• Se genera un reset por un *'opcode ilegal'* cuando la CPU descifra una instrucción que no está definida en el juego de 'opcode'. Ruido excesivo o código que se intenta ejecutar incorrectamente desde el espacio de los datos, pueden causar este evento y pueden devolver a la CPU a un estado conocido.

• Se genera un reset por el módulo *COP,* cuando hay un desbordamiento del contador del COP, indicando que el timer del COP no se reparó a tiempo. Ésta es una protección del sistema contra software no correcto, que devolverá a la CPU a un estado conocido. Para más información véase el módulo COP.

• Se puede generar opcionalmente un reset por módulo *LVI*, cuando V_{DD} cae por dejado de un valor del punto de disparo. Esta característica protege contra el funcionamiento incorrecto de la MCU durante condiciones de bajadas de tensión ('brown-outs'). Para el reset LVI, la línea de reset permanece en estado bajo durante 4096 ciclos de reloj CGMXCLK, después de que V_{DD} se restablece, permitiendo así estabilizar el reloj.

• Power-On Reset (POR) es un reset interno causado por una transición positiva desde 0, en el pin V_{DD}. Todos los relojes internos de la CPU y todos los módulos de la MCU se mantienen inactivos durante 4096 ciclos de reloj CGMXCLK, para así permitir estabilizar el oscilador. Durante este tiempo, el pin RST permanece en estado bajo. El reset POR sólo se puede activar cuando V_{DD} cae a 0 voltios. El POR no es un detector de bajadas de tensión ('brown-out'), detector de bajo voltaje o detector de 'glitch'.

Registro de estado del reset SIM

El registro de estado del reset SRSR del SIM, contiene seis indicadores que muestran la fuente del último reset.

SRSR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	POR	PIN	COP	ILOP	ILAD	0	LVI	0
Escribir:								
POR:	1	0	0	0	0	0	0	0

• Si el *bit POR* del Power-On Reset se pone a 1, el último reset fue causado por el circuito POR.

• Si el *bit PIN* del pin de reset externo se pone a 1, el último reset fue causado por un dispositivo externo poniendo el pin de reset a un nivel bajo.

• Si el *bit COP* se pone a 1, el último reset fue causado por el contador del COP, que cuenta antes de que le fuera dado servicio.

• Si el *bit ILOP* de 'opcode' ilegal se pone a 1, el último de reset fue causado por un 'opcode' ilegal.

• Si el *bit ILAD* de dirección ilegal se pone a 1, el último reset fue causado por un 'opcode' recuperado de una dirección ilegal.

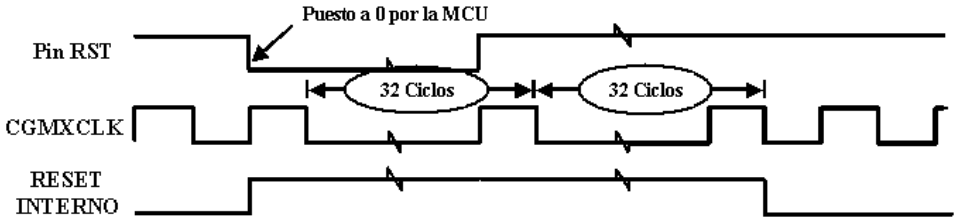
• Si el *bit LVI* de voltaje bajo se pone a 1, el último reset fue causado porque el LVI detectó un voltaje por debajo del punto disparo seleccionado.

Los indicadores (‘flags’) en el registro de estado del reset del SIM se borran cuando se lee este registro. Si este registro de estado del reset del SIM no se ha leído después del varios resets, tendrá señalizados estos resets. Entonces este registro puede ayudar a depurar problemas de código. Por ejemplo, si la aplicación hace un reset involuntariamente, se puede determinar la fuente de reset y corregir la acción tomada.

En caso de un Reset en la aplicación, el registro SRSR ayuda a determinar qué acción tomar. La aplicación puede necesitar de una nueva reestructuración y de una inicialización por un reset del POR que no se necesita con un reset del COP. Otras fuentes de reset pueden necesitar diagnósticos o servicios únicos.

Tiempos del Reset Interno

Ahora que se han visto los diferentes tipos de reset interno, se pueden ver los tiempos internos del reset. Todos los tipos de reset interno, excepto el POR, ponen el pin RST a un nivel bajo durante 32 ciclos de reloj CGMXCLK, para permitir hacer un reset a los dispositivos externos.



La MCU se mantiene en reset durante 32 ciclos de reloj CGMXCLK después de que vuelva a un nivel alto el pin RST, para permitir estabilizar los dispositivos externos. El pin RST se prueba para verificar si el pin RST todavía está en estado bajo. En ese caso, indica que ha ocurrido un reset externo. Si no, indica que ha ocurrido un reset interno y el bit de reset interno se pone a 1, en el registro de estado del reset del SIM.

Tipos de Interrupción

- **Hardware:** Se inician por un pin hardware, que puede ser el pin IRQ o por los puertos de Entrada/Salida, por los puertos serie SCI/SPI o por el módulo TIM. Usan un vector de interrupción y una rutina de servicio. Se pueden enmascarar.
- **Software:** Se ejecutan como parte del flujo de instrucción. Se procesan como una interrupción hardware. No se puede enmascarar.

Hay dos tipos de interrupciones, *hardware* y *software*. Una interrupción no detiene la MCU o el funcionamiento de la instrucción que se está ejecutando, sino que vectoriza el contador de programa a una rutina de servicio de interrupción. Cuando ocurre un evento de interrupción, la MCU primero completa la instrucción en curso y entonces cambia la secuencia de ejecución del programa para responder al evento.

Una interrupción es similar a un reset en que provoca a la MCU recuperar una nueva dirección para el contador del programa y pone a 1 el bit de máscara de interrupción (bit I). Al contrario del reset, las interrupciones sólo suspenden temporalmente la ejecución normal del programa para que el procesador pueda dar servicio a la interrupción. Después de que se ha dado servicio a la interrupción, el procesador vuelve de donde salió para ejecutar el código normal del programa.

Proceso de una Interrupción

Se genera una *interrupción hardware* por una condición interna o externa del hardware y se puede iniciar por un pin hardware o por un módulo de la MCU. Cuando ocurre una interrupción hardware, el contexto del programa se apila (en el ‘stack’), el bit I del registro de código de condición se pone a 1 y se recupera el vector de interrupción. Las interrupciones hardware se pueden. Esto significa que estas interrupciones del hardware sólo se pueden reconocer cuando el bit I se pone a 0.

Se genera una *interrupción software* como resultado de la instrucción SWI. Una interrupción software siempre se ejecuta como parte del flujo de la instrucción. La diferencia importante entre las interrupciones software y hardware, es que las interrupciones software no se pueden enmascarar. Esto significa que el valor del bit I no tiene efecto en interrupciones software. Por otra parte, una interrupción software se procesa de la misma manera que una interrupción hardware.

Fuentes de Interrupción Hardware

El pin IRQ se puede usar para activar interrupciones de hardware externas. Dependiendo de la configuración de la MCU, una interrupción por IRQ se genera por un nivel bajo o por una transición de nivel alto a un nivel bajo en el pin IRQ. Este tipo de interrupción se puede usar para supervisar sistemas externos o eventos.

En la mayoría de las MCU del 68HC08, también se puede generar una interrupción usando pines adicionales del puerto I/O. Estos pins se les denominan KBI (interrupciones de teclado) y normalmente se usan de las entradas de una matriz de teclas. Estas entradas tienen 'pullups' programables que generan una interrupción cuando pasan a un nivel bajo.

Por ejemplo, una matriz de 16 teclas se organiza normalmente como 4 filas y 4 columnas. Las cuatro filas se pueden conectar a las entradas de KBI. Cuando se pulsa una tecla, la entrada de la fila correspondiente se pone a un nivel bajo y se genera una interrupción sin ninguna circuitería lógica adicional. La rutina de servicio de interrupción detecta los rebotes de la tecla y determina qué tecla fue pulsada examinando las columnas.

El ‘timer’ de 16-bits del Módulo TIM puede generar varias interrupciones diferentes que dependen del modelo en particular. El 'output compare', el 'input capture' y las funciones de desbordamiento del ‘timer’ pueden generar interrupciones. Algunos modelos también tienen una característica de interrupción en tiempo real. Se pueden usar estos tipos de interrupciones para procesar eventos basados en una referencia de tiempo.

Para MCUs que tienen un Módulo SCI o SPI, los puertos serie pueden generar una variedad de interrupciones que dependen del tipo. Las interrupciones SCI y SPI incluyen un registro de receptor lleno, un registro de transmisión vacío y un registro de transmisión completa. Se pueden usar estos tipos de interrupciones para procesar eventos de comunicaciones de serie. Otro periféricos como el CAN y el USB también puede generar interrupciones.

Fuentes de Interrupción

Las fuentes de interrupción usan una dirección del vector y una prioridad. La tabla resume las 16 diferentes fuentes de interrupción asociadas a la MCU del 68HC908GP32. La arquitectura del 68HC08 puede manejar hasta 128 diferentes fuentes de interrupción y ‘reset’. Para más información sobre las aplicaciones de la interrupción para cada 68HC08, hay que verificar el libro de datos técnicos de cada dispositivo.

FUENTE	Direcc. Vector	Flag	Máscara	INT	Prioridad
TimeBase	$FFDC - $FFDD	TBIF	TBIE	IF16	16
ADC Conv. Completa	$FFDE - $FFDF	COCO	AIEN	IF15	15
Pin teclado KBD	$FFE0 - $FFE1	KEYF	IMASKK	IF14	14
SCI Trans. Completa	$FFE2 - $FFE3	TC	TCIE	IF13	13
SCI Trans. Vacía		SCTE	SCTIE		
SCI Entrada desocupada	$FFE4 - $FFE5	IDLE	ILIE	IF12	12
SCI Recepción completa		SCRF	SCRIE		
SCI Recepción excedida	$FFE6 - $FFE7	OR	ORIE	IF11	11
SCI Flag de Ruido		NF	NEIE		
SCI Error de cuadro		PE	FEIE		
SCI Error de Prioridad		PE	PEIE		
SPI Transmisión Vacía	$FFE8 - $FFE9	SPTE	SPTIE	IF10	10
SPI Recepción completa	$FFEA - $FFEB	SPFR	SPRIE	IF9	9
SPI Desbordamiento		OVRF	ERRIE		
SPI Modo Fallo		MODF	ERRIE		
TIM2 Desbordamiento	$FFEC - $FFED	TOF	TOIE	IF8	8
TIM1 Canal 1	$FFEE - $FFEF	CH1F	CH1IE	IF7	7
TIM2 Canal 0	$FFF0 - $FFF1	CH0F	CH0IE	IF6	6
TIM 1 Desbordamiento	$FFF2 - $FFF3	TOF	TPOIE	IF5	5
TIM 1 Canal 1	$FFF4 - $FFF5	CH1F	CH1IE	IF4	4
TIM1 Canal 0	$FFF6 - $ FFF7	CH0F	CH0IE	IF3	3
PLL	$FFF8 - $FFF9	PLLF	PLLIE	IF2	2
IRQ	$FFFA - $FFFB	IRQF	IMASK1	IF1	1
SWI	$FFFC - $FFFD	No	No	No	0
Reset	$FFFD - $FFFF	No	No	No	0

Cada fuente de interrupción tiene su propia y única dirección de vector. Esto puede eliminar la necesidad de que el software consulte dentro de una rutina de servicio de interrupción para determinar la fuente correcta de interrupción y producir un servicio de interrupción más rápido.

Cada fuente de interrupción también tiene su propio y único indicador en el registro de estado de interrupción que se puede consultar. Se puede desactivar una fuente de interrupción por un borrado del bit de la única fuente de interrupción.

Cada tipo de interrupción tiene una prioridad predefinida asociada. La interrupción del módulo TBM tiene la prioridad más baja y una IRQ externa tiene la prioridad más alta. Cuando ocurren múltiples interrupciones, la CPU primero mira la prioridad de los eventos y servicios del evento con la prioridad más alta, con los otros pendientes. Un evento de ‘reset’ tiene la prioridad por encima de todos los eventos de interrupción.

El bit I de máscara de interrupción global, activa o desactiva toda interrupción que se procesa con la excepción de interrupción software SWI. Los eventos de interrupción generados por módulos periféricos internos se pueden enmascarar y sólo se reconocen si se pone a 0 el bit I. Estos periféricos tienen típicamente máscaras locales para activar o desactivar tipos específicos de interrupciones.

Por ejemplo, el Módulo TIM tiene una máscara de interrupción separada y el bit se activa por desbordamiento y por cada canal del 'timer'. Los 'resets' no se pueden enmascarar, pero algunos módulos internos se pueden desactivar para que no puedan generar un 'reset'. Los módulos COP y LVI son ejemplos de fuentes potenciales de reset que se pueden desactivar fuera de un 'reset'.

Contexto de Intercambio

Seguidamente se puede ver cómo el Módulo SIM usa esta información para abastecer las interrupciones.

Se denomina proceso de excepción el que determina qué tipo de gestion se necesita. El proceso de excepción a veces se maneja a través de tareas discretas llamadas "contexto que cambia". El proceso de excepción es diferente para el reset y las interrupciones. Sin embargo, las tareas de proceso son las mismas.

Primero, el módulo SIM reconoce los eventos y realiza el arbitraje. Primero se procesa el evento de prioridad más alto. Antes de que se saque el vector, el bit I se pone a 1 para prevenir futuros eventos de interrupción y el contexto actual de la CPU se guarda en el 'stack'. Finalmente se ejecuta la rutina de servicio de interrupción o el manejo de la excepción.

Reconocimiento del Reset y de las Interrupciones

Durante la fase de reconocimiento, se reconocen los resets y se actúa inmediatamente. Las interrupciones se reconocen durante el último ciclo de ejecución de la instrucción. El tiempo de reconocimiento de la interrupción depende de cuando ocurre la interrupción. Si ocurre una interrupción antes del último ciclo de la instrucción en curso, se reconocerá durante el último ciclo y entonces se actuará. Si ocurre una interrupción durante el último ciclo de la instrucción en curso, no se reconocerá hasta el último ciclo de la siguiente instrucción.

Arbitraje

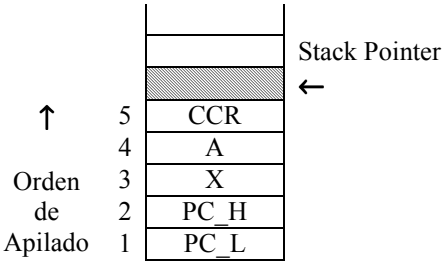
Durante la fase del arbitraje, el Módulo SIM determina la fuente de la excepción y prioriza los eventos de la interrupción para procesar. Todos los resets son considerados iguales y tiene la prioridad más alta por encima de todas las fuentes de interrupción. En el caso de los resets, no se requiere ningún arbitraje. Cualquier reset que ocurra, se actuará inmediatamente y se reflejará en el registro de estado del reset del SIM.

Las interrupciones se arbitran usando niveles de prioridad pre-definidos asociados con las diferentes fuentes de interrupción. El SIM proporciona a la CPU la información sobre que fuentes generaron las peticiones de la excepción. El 68HC908GP32 tienen un total de 16 posibles fuentes de interrupción asociadas con los diferentes módulos internos.

Si la máscara de interrupción se pone a 0, la CPU verificará todas las interrupciones pendientes después de cada instrucción. Si está pendiente más de una interrupción cuando se completa una instrucción, la interrupción de prioridad más alta se abastecerá primero. Una vez se reconoce una interrupción como la interrupción de prioridad más alta, ninguna nueva interrupción se puede tomar antes sin tener en cuenta su prioridad.

Apilado

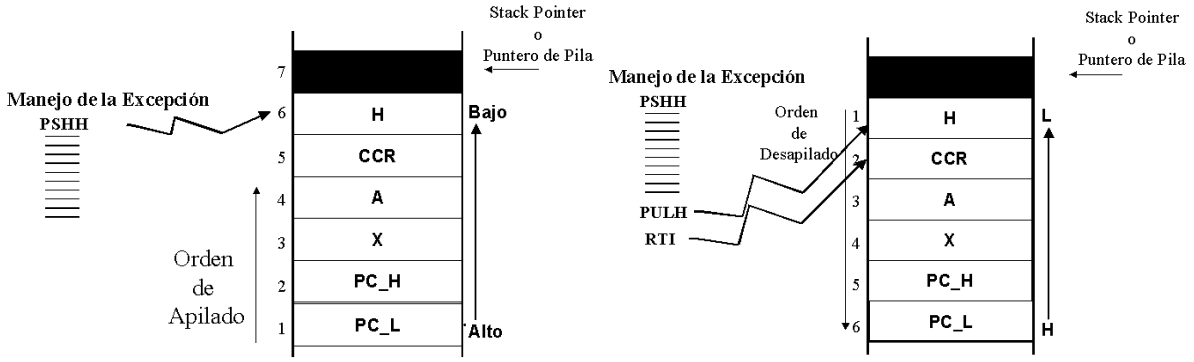
Una vez la MCU ha identificado qué interrupción abastecer, pone a 1 el bit I para prevenir eventos adicionales de interrupción mientras guarda el contexto. En la fase de apilado, se guarda la información crítica de la CPU en el 'stack'. Todos los registros de la CPU se apilan, incluyendo el PC, X, A y el CCR. El registro H no se apila por razones de compatibilidad con el 68HC05. El puntero de pila siempre apunta al siguiente byte disponible en la pila.



Debido a que un evento de reset causa restablecer la CPU y el 'stack', el apilado no se realiza para eventos de reset. Después de que todos los registros de la CPU se han guardado en el 'stack', el contador de programa se carga con una dirección del vector definida por el usuario y empieza el proceso. Esta dirección de 16-bits es la dirección de inicio para la rutina de servicio de interrupción o manipulador de la excepción.

Manipulación de una Excepción

Véase el orden de apilado de los registros. Los registros se apilan de la dirección más alta a la más baja. Antes se mencionó que el registro H no se guarda automáticamente por la CPU. Por consiguiente, debe guardarse al principio del manejo de la excepción, ejecutando una instrucción PSHH. Esto significa que se tiene que restaurar antes de terminar el manejo de la excepción.



Antes de que se restaure el contexto viejo, véase el orden para desapilado. Se empieza ejecutando la instrucción PULH para restaurar el registro H. Luego, se ejecuta la instrucción RTI para restaurar los registros restantes y se empieza con el registro CCR. Cuando el registro CCR se restaura, se borra a la condición original, el bit de máscara de interrupción global también se restaura y activa las interrupciones. Cuando el contexto viejo se restaura, continúa el funcionamiento normal del programa de la aplicación. Si hay interrupciones adicionales pendientes, el proceso empezará de nuevo.

Ejemplo para Atrapar Interrupciones no usadas

Un método preventivo tolerante a fallos para manejar vectores de interrupción sin utilizar, es usar una trampa con 'wachdog timer' del COP. Con el módulo COP activado, este método usa un bucle para atrapar interrupciones no deseadas. El COP se restablecerá después de 'timeout'. Esto proporciona una manera de restablecer la MCU cuando ocurre una condición inesperada o de una interrupción falsa.

```
                                ;* Uso de una interrupción "Trap" con el Watchdog del COP
                                ;* Vectores no usados
TRAP:       bra     TRAP        ; espera un reset del COP
            org     $1FF8       ; Vector del Timer
            fdb     TRAP        ; Puntos a atrapar
            org     $1FFC       ; Interrupciones Software
            fdb     TRAP        ; Puntos a atrapar
```

En algunos 68HC08, los valores del vector definidos por el usuario se usan como seguridad. Los valores de dirección idénticos asignados a las interrupciones sin usar, no pueden ser aceptados. Si ésta es una consideración, se puede duplicar el bucle cerrado "trampa" en posiciones de memoria diferentes para mejorar la seguridad.

Módulo COP (Computer Operating Properly)

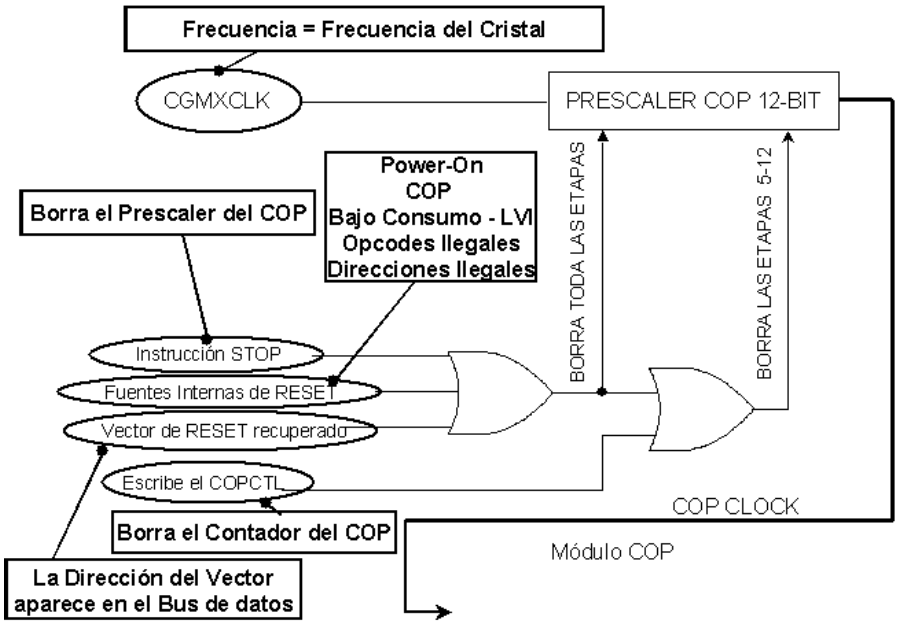
En esta parte se describen las características y la configuración del módulo COP. El 'timer' COP, también conocido como 'Watchdog Timer' se basa en un contador que corre libremente y que se puede borrar por el código del usuario. El COP permite a la CPU recuperarse de eventos inesperados como el llamado 'runaway software' (ejecución incorrecta del software) y errores en el proceso de software.

Para usar el COP, simplemente hay que activar y seleccionar el periodo de 'timeout' deseado. El COP hace un 'reset' si no ha habido un Reset dentro del periodo de 'timeout'. El 'timeout' es el exceso de tiempo en la espera de una señal determinada. Lo más importante que siempre se debe recordar, es que hay que servirlo antes de que el periodo de interrupción expire. Si se activa el 'timer' del COP y no se le da servicio dentro del periodo de interrupción, el COP hará un reset a la CPU.

Seguidamente se verán las diferentes señales y bloques asociados al módulo COP, empezando con el prescaler del COP.

Prescaler del COP

Partiendo de la señal CGMXCLK, que es la señal de salida del oscilador a cristal, cuya frecuencia CGMXCLK es igual a la frecuencia del cristal, entra al prescaler de 12 bits y la señal de salida es el reloj COP.



La instrucción STOP borra automáticamente el prescaler del COP. Esto se puede evitar, desconectando inadvertidamente el COP, con una instrucción STOP para desactivar la instrucción STOP. Cuando se desactiva la instrucción STOP y se ejecuta una instrucción STOP resulta un 'reset' por un 'opcode' ilegal.

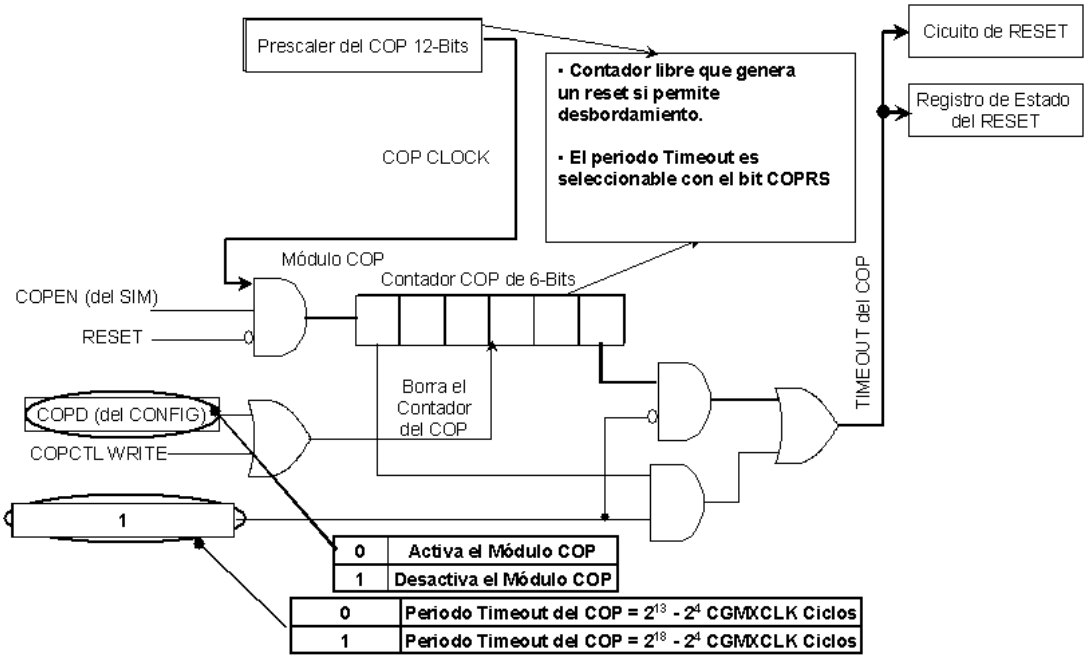
Las fuentes de 'reset' internas por Power-On, por el módulo COP, por el módulo LVI, por un 'opcode' ilegal o por una dirección ilegal, borrarán el prescaler y el contador del COP.

Ocurre la recuperación de un vector de reset, cuando la dirección del vector aparece en el bus de datos. Un vector de reset recuperado borra el prescaler del COP.

Escribiendo algún valor en el registro de control COPCTL del COP, se borra el contador del COP y se borran los bits de las etapas 5 al 12 del prescaler.

Diagrama de Bloques del COP

La siguiente figura muestra las señales restantes del COP y sus bloques asociados. El COP se activa fuera del reset. Si no se desea el funcionamiento del COP se debe desactivar inmediatamente poniendo a 0 el bit COPD del registro de configuración del COP. El registro de configuración es un registro que se escribe una sola vez, para proteger contra código 'runaway' que puede desactivar el COP inadvertidamente. Una vez se escribe el registro CONFIG con el COP activado, el COP no se puede desactivar hasta que se haga un Reset en el dispositivo.



El bit COPRS de selección de velocidad del COP en el registro de configuración, selecciona uno de los dos periodos de 'time out'. Poniendo a 1 el bit COPRS se selecciona un periodo de 'time out' del COP de $2^{13} - 2^4$ ciclos CGMXCLK. Con el bit COPRS puesto a 1, un cristal de 32.768-kHz da un periodo de 'timeout' de 250 ms del COP. Poniendo a 0 el bit COPRS, se selecciona un periodo de 'timeout' del COP de $2^{18} - 2^4$ ciclos CGMXCLK.

El contador del COP, es un contador de 6 bits que cuenta libremente, precedido por un contador de 12 bits del prescaler. Si no se borra por software antes de que expire el periodo de 'timeout', los desbordamientos del contador del COP y genera un reset asíncrono. Para prevenir un reset del COP, se escribe algún valor en el registro de control COPCTL del COP antes de que ocurra un desbordamiento. Esto borra el contador del COP y las etapas del 5 al 12 del prescaler.

Ejemplo de Velocidad de Refresco del Reset

En este ejemplo se puede ver cómo calcular el periodo de 'timeout' del COP. Para calcular el periodo de 'timeout' del COP en segundos, hay que multiplicar los valores de los 12 bits del prescaler del COP y el contador de 6 bits del COP, dividiendo el resultado por la frecuencia de CHMXCLK.

Para este ejemplo, se pone a 1 el bit COPRS. Este selecciona un periodo de 'timeout' del COP de 218 - 24 ciclos de CGMXCLK. Con el bit COPRS puesto a 1, los 12 bits del prescaler del COP contarán 213 o 8192 ciclos. El contador de 6 bits del COP contará 25 o 32 ciclos.

La frecuencia CGMXCLK es igual que la frecuencia del cristal. Para este ejemplo, se asume una frecuencia del cristal de 4 MHz. Multiplicando 8192 por 32 y dividiendo el resultado por 4 MHz, da un periodo de 'timeout' de 65.53 ms. Esto representa el periodo, en segundos, que el COP debe abastecer para evitar un reset del COP.

La Velocidad de Refresco del Reset depende de la frecuencia CGMXCLK. El COP puede ser abastecido dentro de:

Prescaler del COP de 12 bits = $2^{13} = 8191$
Prescaler del COP de 5 bits = $2^5 = 32$

$$\frac{8191*32}{\text{FrecuenciaCGMXCLK}} \text{segundos}$$

Ejemplo: Cristal de 4 MHz externo (CGMXCLK = 4 MHz)

$$\frac{8191*32}{4000000} \approx 0.065536 segundos \approx 65.53 ms$$

Registro de Control del COP

El registro de control COPCTL del COP, se localiza en la dirección $FFFF y se solapa con el vector de reset de la CPU. Leyendo la posición $FFFF devuelve el byte bajo del vector de reset de la CPU. Escribiendo algún valor en la dirección $FFFF borra el contador del COP y el contador del SIM. Esto previene un reset del COP y empieza un nuevo periodo de 'timeout'.

COPCTL:

$FFFF	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	Byte bajo del Vector de Reset							
Escribir:	Borra el contador del COP							
Reset:	No le afecta el Reset							

Método Protección del Sistema

- Usa una instrucción o una sección del código para refrescar el COP.
- Incluye instrucciones de refresco en el bucle principal del código.
- Hay que evitar usar en subrutinas de temporización o de interrupción.
- Hay que abastecer el COP dentro de un periodo mínimo de 'timeout'.

Ahora se pueden ver algunos métodos que se pueden usar para proteger el sistema del software 'runaway'.

Para proteger el sistema del 'runaway software', se debe usar una instrucción o una sección de código para refrescar el COP. En una situación de 'runaway', el software se puede coger en un bucle del código que refresca el COP. Si esto pasa, el COP se refrescará cuando el código no se está ejecutando correctamente. En este caso el dispositivo no hará un reset.

En algunos casos, no se puede evitar usar instrucciones de múltiples refrescos a lo largo del código. Un ejemplo es cuando hay una posibilidad de que el refresco no puede ocurrir dentro del periodo de 'timeout', incluso cuando el código se está ejecutando correctamente. Si es necesario usar instrucciones de múltiples refrescos, hay que intentar poner el número de instrucciones a un mínimo.

Las instrucciones de refresco deben estar en el bucle principal del código. Cuando las instrucciones de refresco están en el bucle principal del código y los funcionamientos del dispositivo se supervisan correctamente, se puede estar seguro que el COP se está refrescando correctamente.

Hay que evitar usar instrucciones de refresco en subrutinas. Sin embargo es improbable que la subrutina sea llamada por el código 'runaway', sobre todo si la dirección de la subrutina del watchdog está en la pila de la RAM. También, hay que evitar usar instrucciones de refresco en un timer o en otras rutinas de interrupción. Aunque pueda parecer conveniente refrescar el COP en una rutina de interrupción del timer, no es una buena práctica y se debe evitar siempre que sea posible.

Puede haber casos en que las interrupciones son la única posibilidad de refrescar el COP. Un ejemplo de esto, es cuando el dispositivo se pone en modo 'wait' para reducir el consumo de energía. En este caso, se puede usar otra rutina que pone a 1 los bits en la RAM cuando estas rutinas están operando correctamente. Estos bits se pueden probar en la rutina de interrupción, proporcionando algún nivel de seguridad de funcionamiento correcto. Si los bits son incorrectos, la acción de recuperación necesaria se puede llevar a cabo.

Siempre se debe considerar el periodo de 'timeout' mínimo para el dispositivo. Si no se abastece el COP dentro del periodo mínimo de 'timeout', se hará un reset al dispositivo si se activa el COP. El periodo mínimo de 'timeout' se proporciona en el libro de datos técnicos del dispositivo.

Módulo LVI (Low Voltaje Inhibit)

En esta parte se explica como configurar y usar el Módulo LVI para proteger el sistema durante una caída de alimentación, haciendo un reset a la MCU. Cuando el voltaje vuelve a su nivel nominal, la MCU continuará trabajando. Se puede seleccionar el voltaje de disparo para trabajar a 3V y 5V. Este módulo reduce el número de componentes y el costo del sistema.

Usos y Características del LVI

- Mejora la fiabilidad del sistema.
- Reduce el número de componentes externos y bajando el costo.
- Hace un reset a la MCU cuando el voltaje baja a partir de un nivel.
- Cuando el voltaje vuelve a su nivel nominal, la MCU continúará trabajando.
- Incluye una selección del voltaje de disparo para sistemas a 3V y 5V.

El funcionamiento de un sistema puede ser afectado por la caída del voltaje de alimentación. Los problemas de alimentación son causas comunes de una condición de 'brown-out' (caída de la fuente de alimentación) o de desgaste de una batería que alimenta al sistema. El Módulo LVI protegerá el trabajo de la MCU durante estos eventos.

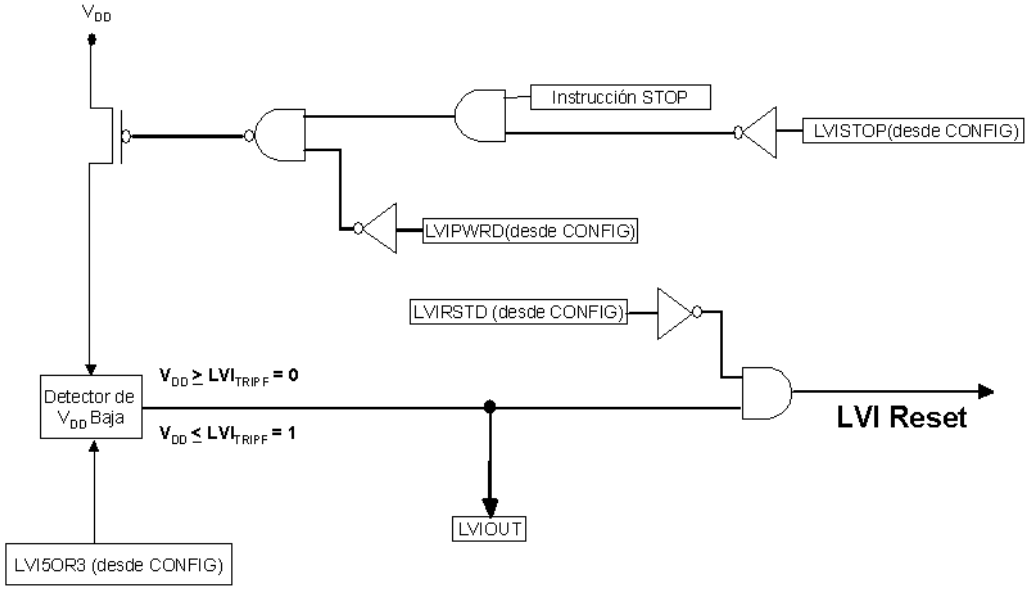
Los circuitos externos LVI tienen un costo típico de más de 0.30$, integrando esta función en el microcontrolador, se reduce el costo del sistema.

Cuando el circuito LVI detecta una caída de voltaje, puede hacer un reset a la CPU evitando una conducta errónea del proceso. Cuando el voltaje vuelve a ser el apropiado, la CPU continúa el proceso. Esto ayuda a evitar errores del sistema.

La mayoría de MCU de la familia 68HC08 permiten al usuario seleccionar uno de los dos puntos de disparo del LVI, 5V y 3V. Para más información sobre los puntos de disparo específicos para cada MCU en particular, véase la sección de características técnicas eléctricas de cada MCU en el libro de datos técnicos.

Diagrama de Bloques del LVI

Ahora, se puede ver cómo se configura este Módulo LVI. El Módulo LVI contiene un circuito de referencia 'bandgap' y un comparador que determinan cuando la MCU opera con un voltaje por debajo del punto de disparo especificado. El Módulo LVI se activa fuera del reset y se configura usando varios bits en el registro CONFIG de configuración del sistema. Este registro se inicializa típicamente con el 'Power-On Reset'. Una vez se ha escrito el registro no se puede escribir de nuevo hasta que ocurra el siguiente reset. Esta característica de 'escribir una sola vez' asegura que una aplicación no puede reconfigurar la MCU inadvertidamente.



• El *bit LVIPWRD* desactiva la alimentación del LVI, también permite al Módulo LVI supervisar el voltaje de alimentación (V_{DD}). Poniendo este bit a 0, se aplica alimentación al circuito analógico LVI. Poniendo a 1 el bit LVIPWRD, se quita la alimentación del Módulo LVI.

• El *bit LVISTOP* activa el modo 'stop' del LVI, determina si el LVI opera en modo 'stop'. Cuando el bit LVIPWRD se pone a 0 y poniendo el bit LVISTOP a 1, permite al Módulo LVI operar durante el modo 'stop'. Un reset borra el bit LVISTOP. La habilidad de desactivar el LVI automáticamente en modo 'stop' permite

reducir el consumo del circuito y alarga la vida de la batería, mientras protege a la CPU durante el funcionamiento normal.

• El *bit LVI5OR3* selecciona el modo de voltaje en que trabaja el Módulo LVI, a 5 V o 3 V. El modo de voltaje en que trabaja el LVI, debe ser igual al voltaje de alimentación (V_{DD}). Poniendo a 1 el bit LVI5OR3, configura el punto de disparo (V_{TRIPF}) para un funcionamiento nominal de 5V. Borrando el bit LVI5OR3, configura el punto de disparo (V_{TRIPF}) para un funcionamiento nominal de 3V. En reset, el LVI tiene un valor predefinido de 3V. Si alimenta un sistema a 5V, el bit LVI5OR3 típicamente hay que ponerlo a 1 para elevar el punto de disparo para trabajar a 5V, después de cada 'power-on reset'. Los valores del punto de disparo están típicamente en el rango de 4.2 V a 4.5 V en el modo de 5V o de 2.4 V a 2.7 V en el modo de 3V. Para la información sobre los valores del punto de disparo específicos, véase las especificaciones eléctricas de cada MCU en el libro de datos técnicos.

• El *bit LVIRSTD* desactiva la señal de reset del Módulo LVI. Cuando se pone el bit LVIRSTD a 0, el Módulo LVI generará un reset cuando LVIOUT se pone a 1, lo que significa que V_{DD} ha caído por debajo del voltaje de disparo V_{TRIPF}. Si ocurre esta condición, la MCU permanecerá en la condición de reset hasta que V_{DD} suba por encima del voltaje de disparo V_{TRIPR}. V_{TRIPF} es más bajo que V_{TRIPR}, por histéresis, típicamente de 0.1 V, para evitar entrar y salir de 'reset' cuando el voltaje de alimentación tiene un ligero rizado cercano al voltaje de disparo.

Registro de estado del LVI (LVISR)

El Módulo LVI usa el registro de estado LVISR, para indicar que el voltaje V_{DD} ha caído por debajo del nivel de V_{TRIPF}. El bit LVIOUT del LVI, es un indicador de estado de sólo lectura, que el Módulo LVI pone a 1 cuando V_{DD} es menor de V_{TRIPF}. El Módulo LVI pone a 0 este bit cuando V_{DD} sube por encima de V_{TRIPR}. Un reset pone a 0 el bit LVIOUT.

LVISR:

$FEOC	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	LVIOUT	0	0	0	0	0	0	0
Escribir:								
Reset:	0	0	0	0	0	0	0	0

Este indicador de estado es particularmente útil cuando el Módulo LVI se usa en modo de funcionamiento en modo consulta, llamado modo 'polled' (no es operativo, solo monitoriza la alimentación y no se puede utilizar para generar un reset). Este modo normalmente se usa en aplicaciones donde se trabaja con niveles de V_{DD} por debajo del nivel de V_{TRIPF}, donde se desea aumentar la vida de la batería, por ejemplo. En este caso, el software puede supervisar V_{DD} consultando ('polling') el indicador de estado LVIOUT.

Para configurar el LVI para este modo, hay que activar el Módulo LVI poniendo a 0 el bit LVIPWRD y a 1 el bit LVIRSTD para desactivar resets del LVI.

Ejemplo de un Reset del LVI

Primero, se pone a 1 el bit LVISTOP. Esto permite al Módulo LVI operar durante una instrucción de STOP. Se alimenta el LVI poniendo a 0 el bit LVIPWRD. Cuando V_{DD} cae por debajo del voltaje de disparo V_{TRIPF}, la lógica del LVI detecta esta condición y pone a 1 el bit LVIOUT. Si el bit LVIRSTD = 0, se enviará un reset a la CPU.

Módulo SPI (Serial Peripheral Interface)

En esta sección se describen las características y la configuración del Módulo SPI, para trabajar en modo maestro (‘master’) o modo esclavo (‘slave’) y se describe el proceso de transmisión de datos. También se escribe un programa para configurar un SPI maestro para la transmisión de datos.

El SPI es un módulo de comunicaciones serie, síncrona y full-duplex. Está pensado principalmente para comunicaciones dentro del mismo circuito impreso a velocidades relativamente altas. El SPI es una buena manera de unir vía serie la MCU con periféricos externos, como: pequeñas eeproms, convertidores analógico digitales de alta resolución, convertidores digitales analógicos, módulos LCD, controladores de visualizadores a LED de siete segmentos, etc. Comunicando vía serie, la MCU y los periféricos necesitan menos pins y esto revierte a encapsulados más pequeños y normalmente de menor costo. También el SPI se puede usar para comunicar con otras MCUs.

Características:

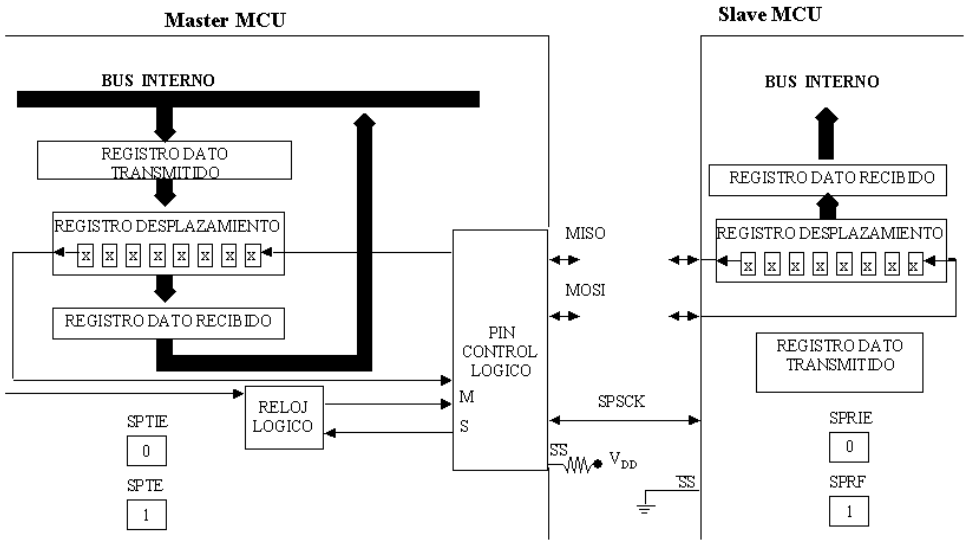
- Máxima frecuencia modo ‘master’ = frecuencia de bus / 2
- Máxima frecuencia modo ‘slave’ = frecuencia de bus
- Reloj Serie con polaridad y fase programable
- Activa dos interrupciones separadas:
  - SPRF (Receptor del SPI Completo)
  - SPTE (Transmisor del SPI Vacío)

El SPI opera en modo maestro (‘master’) o en modo esclavo (‘slave’). En modo ‘master’, el SPI genera un reloj de comunicación síncrona, a una de las cuatro frecuencias posibles del ‘master’. La frecuencia máxima en modo ‘master’ es la mitad de la frecuencia del bus. Para la mayoría de MCUs del 68HC08, la frecuencia máxima del bus es de 8 MHz, permitiendo hasta 4 MHz de reloj en modo ‘master’. En modo ‘eslave’, el SPI puede operar a velocidades de reloj hasta la frecuencia de bus o hasta 8 MHz, en la mayoría de MCUs del 68HC08.

En el SPI también se puede configurar la polaridad y la fase del reloj, permitiendo al SPI comunicar con la mayoría del periféricos serie. Se puede configurar el SPI para generar dos eventos de interrupción separados: transmisor vacío y receptor lleno. Cada interrupción tiene un vector separado que permite transferencias eficaces.

Funcionamiento del SPI

El SPI se ha desarrollado alrededor de un doble registro de desplazamiento 'buffered' de 8 bits con ambos extremos del registro de desplazamiento, que van a los pines de la MCU. Un extremo del registro de desplazamiento se conecta al pin MISO (Master-In Slave-On). Este pin actúa como una entrada para el Módulo ‘master’ del SPI y como una salida para el Módulo ‘slave’ del SPI. El otro extremo del registro de desplazamiento se conecta al pin MOSI (Master-On Slave-In).

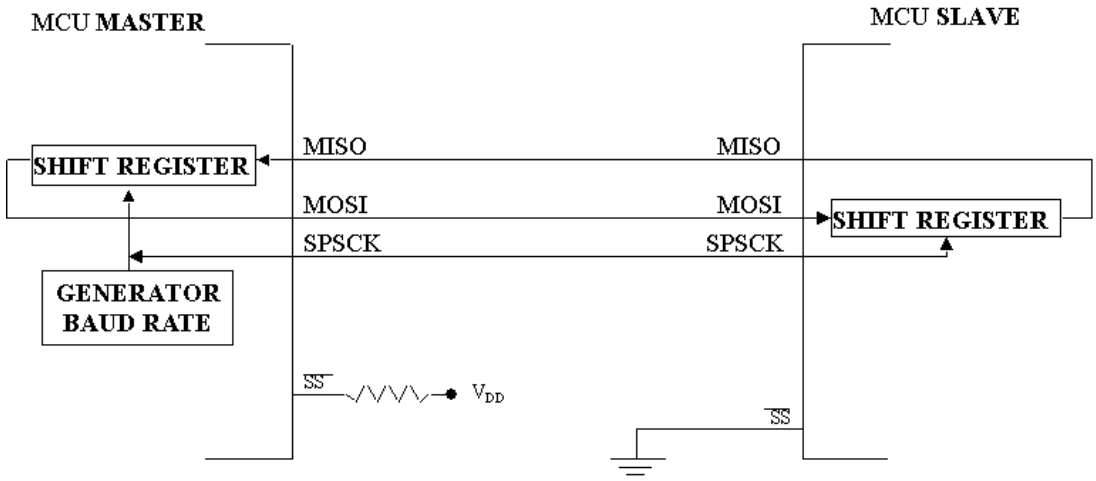


La CPU empieza una transferencia serie SPI, escribiendo un byte de datos en el registro de datos transmitidos. Se transferirán automáticamente vía serie todos los 8 bits de datos a través del pin MOSI del ‘master’, sincronizados con el reloj de salida del ‘master’ (SPSCK). Cada vez que se desplaza un bit a través del

pin MOSI del 'master', se desplaza un bit a través del pin MISO del 'master', permitiendo una comunicación 'full-duplex'. Más adelante se mostrará un ejemplo de trasferencia más detallado.

Conexiones Master-Slave del SPI

En la figura siguiente se pueden ver las conexiones entre un SPI 'Master' y un SPI 'Slave'. Primero se debe configurar el SPI del circuito como 'Master' y el otro circuito como 'Slave'. Para activar el SPI como Master, se une el pin $\overline{SS}$ a V_{DD}. Para activar el SPI como 'Slave', hay que conectar el pin $\overline{SS}$ a masa.



Esto permite a un circuito 'Master' seleccionar con qué circuito 'Slave' se va a comunicar, en el caso de que se necesiten varios circuitos periféricos 'Slave'. Típicamente, un 'Master' usa un pin del puerto de I/O de la MCU para controlar cada pin $\overline{SS}$ de cada 'Slave'. Si el pin $\overline{SS}$ de un 'Slave' se pone a 1, el pin MISO del 'Slave' se pone en un estado de alta impedancia. Cuando ocurre esto, el 'Slave' ignorará todos los relojes SPSCK entrantes, aunque esté en medio de una transmisión. Se puede configurar y supervisar el funcionamiento del SPI usando los tres registros del Módulo SPI.

Registro SPCR de control del SPI

SPCR	Bit 7	6	5	4	3	2	1	Bit 0
Leer: / Escribir:	SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
Reset:	0	0	1	0	1	0	0	0

- El *bit SPRIE* de interrupción del receptor, es un bit de lectura/escritura que activa las peticiones de interrupción. Cuando el bit SPRIE se pone a 1, se genera una interrupción cuando el bit SPRF receptor del SPI se pone a 1. El bit SPRF se pone a 1 cuando se transfiere un byte de datos al registro de datos del SPI.
- El *bit DMAS* de selección de DMA, es un bit de sólo lectura que no tiene efecto en las MCU 68HC08 que no tienen un controlador de DMA.
- El *bit SPMSTR* 'master' del SPI, es un bit de lectura/escritura que selecciona el modo de trabajo del SPI. Poniendo el bit a 1 para seleccionar el modo 'master' y a 0 para seleccionar el modo 'slave'.
- El *bit CPOL* de polaridad del reloj, es un bit de lectura/escritura que determina el estado lógico del bit SPSCK entre las transmisiones. Para transmitir datos entre dos módulos SPI, los módulos SPI deben tener valores idénticos del bit CPOL.
- El *bit CPHA* de la fase del reloj, es un bit de lectura/escritura que controla la relación de tiempo entre el reloj serie y los datos del SPI. Para transmitir datos entre dos módulos SPI, deben tener valores idénticos de CPHA. La fase y la polaridad se podrán ver más adelante.
- El *bit SPWOM* modo OR alambrado, es un bit de lectura/escritura que desactiva los 'pullup' en los pins SPSCK, MOSI y MISO para que estos pins se pongan en salida 'open-drain'.
- El *bit SPE* activa el SPI, es un bit de lectura/escritura que activa el módulo SPI. Se debe desactivar el SPI antes de que cambie la fase o la polaridad del reloj, escribiendo el bit CPOL o el bit CPHA.
- El *bit SPTIE* activa la interrupción de transmisión, es un bit de lectura/escritura que activa las peticiones de interrupción de la CPU que se vayan a generar cuando se transfiere un byte desde el registro transmisor de datos al registro de desplazamiento.

Registro de estado y de control del SPI (SPSCR)

El registro SPSCR de estado y de control del SPI, contiene el indicador de estado y los bits de control adicionales.

SPSCR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	SPRF	ERRIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0
Escribir:								
Reset:	0	0	0	0	1	0	0	0

• El *bit SPRF* de receptor lleno del SPI, es un bit de sólo lectura que se pone a 1 cada vez que transfiere un byte del registro de desplazamiento al registro de datos del receptor. Si el bit SPRIE en el registro SPCR se pone a 1, poniendo a 1 el bit SPRF genera una petición de interrupción a la CPU. Durante una interrupción de la CPU, la CPU pone a 0 el bit SPRF leyendo el estado del SPI y el registro de control, con el bit SPRF puesto a 1 y después leyendo el registro de datos del SPI.

• El *bit ERRIE* activa la interrupción de error, es un bit de lectura/escritura que le permite al SPI generar interrupciones en condiciones de error para tomar acciones correctivas. Las dos condiciones de error son: desbordamiento y modo fallo.

• El *bit OVRF* de desbordamiento, es un bit de lectura/escritura que se pone a 1 cuando el software no lee el byte en el registro de datos del receptor antes de que el siguiente byte completo entre en el registro de desplazamiento. En una condición de desbordamiento, el último byte que se ha desplazado se pierde. Se puede poner a 0 el bit OVRF leyendo el registro de estado y de control con el bit OVRF puesto a 1, leyendo el registro de datos del receptor.

• El *bit MODF* de modo fallo, es un indicador de sólo lectura que se pone a 1 cuando ocurre un error de fallo durante la transmisión y el bit MODFEN que activa el modo de fallo se pone a 1. Las condiciones bajo las que el indicador MODF se pone a 1, son diferentes para el 'master' y el 'slave'. En un SPI 'master', el bit MODF se pone 1 siempre que el pin $\overline{SS}$ pasa a valor bajo. En un SPI 'slave', el bit MODF se pone a 1 cuando el pin $\overline{SS}$ pasa a valor alto, durante la transmisión.

• El *bit SPTE* de transmisor vacío, es un bit de lectura/escritura que se pone a 1 cada vez que el registro de datos de transmisión transfiere un byte al registro de desplazamiento. Poniendo a 1 el bit SPTE, genera una interrupción si es activado por el bit SPTIE. No se debe escribir en el registro de datos del SPI a menos que el bit SPTE se ponga a 1.

• Los *bits SPR1 y SPR0* de selección de 'baud rate', son bits de lectura/escritura que selecciona uno de los cuatro divisores de velocidad de transmisión. Puesto que el reloj sólo se maneja en modo 'master', estos bits no tienen efecto en modo 'slave'.

Cálculo del Baud Rate

La tabla siguiente muestra los códigos para seleccionar uno de los cuatro divisores de 'baud rate', usando los bits SPR1 y SPR0 de selección de la velocidad de transmisión del SPI. La velocidad de transmisión se calcula dividiendo la frecuencia del bus por dos veces el divisor de 'baud rate'.

SPR1 y SPR0	Baud rate Divisor (BD)
00	2
01	8
10	32
11	128

Para calcular la velocidad de transmisión del SPI se usa esta formula:

$$Baud\ Rate = \frac{CGMOUT}{2 \times BD}$$

Donde,

CGMOUT = Reloj Base de salida del módulo generador de reloj (CGM)
BD = División de Baud rate

Por ejemplo, asumiendo una frecuencia de bus de 8 Mhz y poniendo SPR1 y SPR0 a 0, se selecciona el 'baud rate' divisor por 2. La frecuencia de bus es 8 Mhz, dividido por dos veces el divisor de 'baud rate', que es 4, da un baud rate de 2 Mhz.

$$Baud\ Rate = \frac{CGMOUT}{2 \times BD} = \frac{8}{2 \times 2} = \frac{8}{4} = 2MHz$$

Registro SPDR de Datos

El registro SPDR de datos del SPI, realmente tiene dos registros que se acceden con una sola dirección, y son el registro de datos del receptor y el registro de datos del transmisor. Estos registros están separados y contienen valores diferentes. Se accede a los registros individualmente usando las acciones de lectura/escritura. Leyendo el registro de datos, se leen los datos del registro de datos del receptor. Escribiendo el registro de datos del SPI se escriben los datos en el registro de transmisión de datos.

SPDR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	R7	R6	R5	R4	R3	R2	R1	R0
Escribir:	T7	T6	T5	T4	T3	T2	T1	T0
Reset:	No le afecta							

No se deben usar las instrucciones de lectura, de modificación o de escritura en el registro SPDR de datos del SPI en el momento que el registro leído no es igual a registro escrito. En 'reset', el estado de los registros es indefinido.

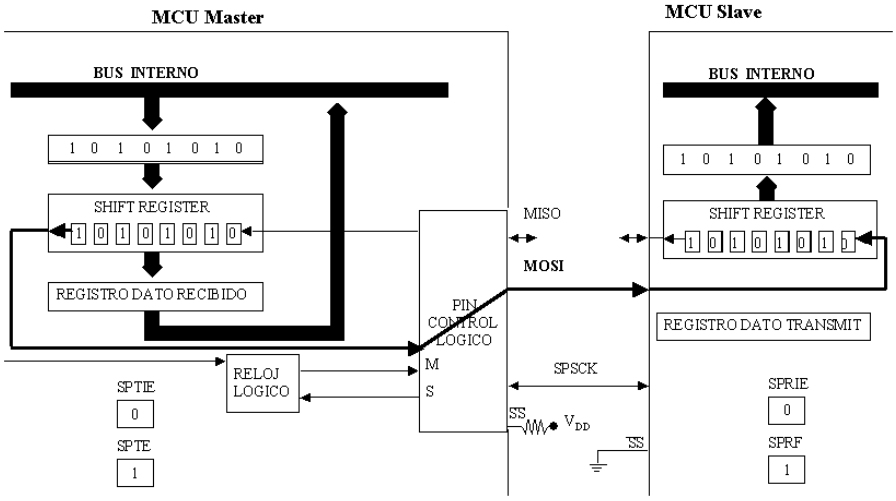
Inicialización del SPI 'Master-Slave'

Paso	**Bits de Control**
1) Inicializar la frecuencia de reloj SPI.	SPR1 y SPR0 en SPSCR
2) Configurar el reloj.	CPOL y CPHA en SPSCR
3) Selección del modo master/slave.	SPMSTR en SPCR
4) Activar la interrupción, si se desea.	SPTIE y SPRIE en SPCR
5) Activar el SPI master.	SPE en SPCR
6) Activar el SPI slave.	SPE en SPCR

Para inicializar el SPI 'master-slave', primero hay que seleccionar la frecuencia de reloj usando los bits SPR1-SPR0 de selección de 'baud rate'. Después hay que configurar el reloj con el bit CPOL de polaridad del reloj y el bit CPHA de fase del reloj. Seguidamente se selecciona el modo de SPI, usando el bit SPMSTR 'master del SPI. Poniendo el bit a 1 para el SPI 'master' y 0 para el SPI 'slave'. Si se desea que actúe el funcionamiento de interrupción, hay que activar las interrupciones usando el bit SPTIE de activación de la interrupción del transmisor del SPI y el bit SPRIE de activación de la interrupción del receptor. Finalmente, activar el bit del sistema SPI, usando el bit SPE que activa el SPI, en el registro de control, asegurándose que el SPI 'master' se activa antes que el SPI 'slave'.

Ejemplo de transmisión de datos master-slave

El software empieza una transmisión SPI, escribiendo un byte en el registro de datos de transmisión del SPI. Si el registro de desplazamiento está vacío, el byte se transfiere inmediatamente a este registro de desplazamiento. Por otra parte, la transferencia empieza cuando el registro de desplazamiento termina de transferir el byte anterior. Una vez el byte es transferido desde el registro de transmisión del SPI al registro de desplazamiento, el bit SPTE se pone a 1 indicando que se puede escribir otro byte al registro de transmisión de datos del SPI. El bit SPTE genera una interrupción a la CPU, si el bit SPTIE se pone a 1. Esto permite manejar la interrupción de transmisión de transferencia multi-byte. En este ejemplo, el bit SPTIE se pone a 0, para desactivar las interrupciones.



El byte empieza desplazando un bit, en el pin MOSI sincronizado con la señal SPSCK de reloj 'master'. La transferencia continuará durante 8 SPSCK ciclos de reloj y transferirá todos los 8-bits. Como que el byte se

ha desplazado fuera del SPI 'master', se desplaza otro byte en el pin MISO del 'slave'. En este ejemplo, se desatiende el byte de información que se desplaza del SPI 'slave' al SPI 'master'.

La transmisión acaba cuando el byte completo se desplaza fuera del registro de desplazamiento del SPI 'master' y en el registro de desplazamiento del SPI. El registro de desplazamiento 'slave' se transfiere automáticamente al registro de datos del SPI 'slave' si está vacío y permite transferir otro byte si hay alguno pendiente. El bit SPRF se pone a 1 indicando que el registro SPI receptor está lleno y esperando ser leído. Si el bit SPRIE se pone a 1, se genera también una petición de interrupción del SPI receptor. En este ejemplo, el bit SPRIE se pone a 0.

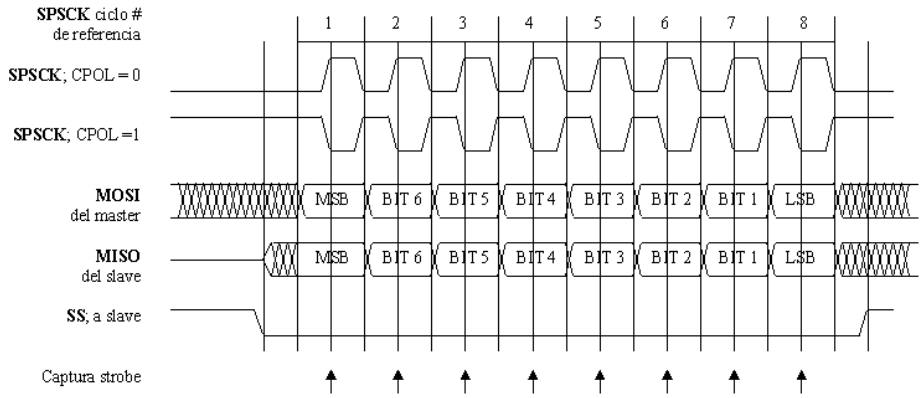
Para evitar un desbordamiento, el 'slave' debe leer el registro de datos del receptor anterior para intentar que el 'master' transfiera más de un byte adicional. Por ejemplo, si el 'master' ha transmitido con éxito un byte que está esperando ser leído en el registro de datos del receptor, el 'master' puede transferir un byte adicional. Este byte adicional se mantendrá en el registro de desplazamiento y no se cargará en el registro del receptor hasta que el byte anterior no haya sido leído por la CPU. Si el 'master' intenta transmitir vía serie otro byte antes de leer el registro de datos del receptor, se sobrescribirán los datos en el registro de desplazamiento 'slave' causando una condición de error por desbordamiento. El byte en el registro de datos del receptor no se sobrescribirá.

Formatos de Transmisión

El software puede seleccionar cualquiera de las cuatro combinaciones de la fase del reloj y polaridad del SPI, usando los bits CPOL y CPHA. La polaridad del reloj selecciona un reloj alto o bajo activo y no tiene efecto significante en el formato de la transmisión. La fase del reloj selecciona uno de dos formatos de la transmisión fundamentalmente diferentes.

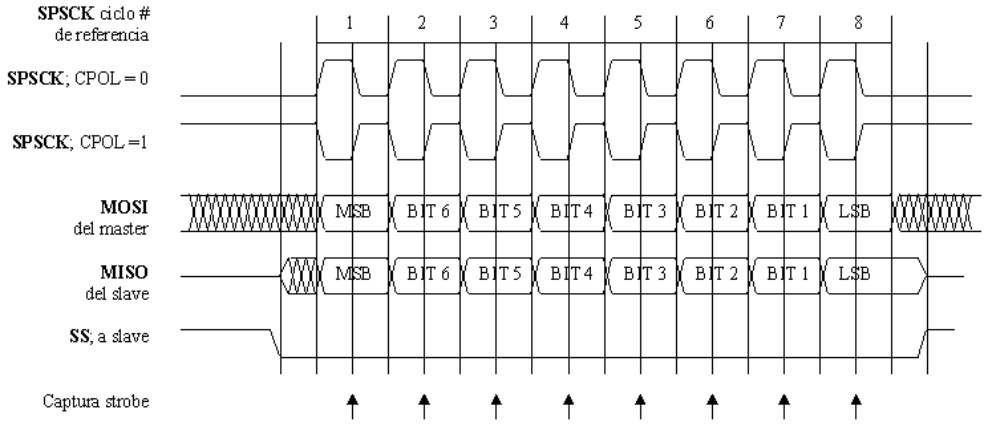
Seguidamente se examina la transmisión SPI para cada fase del reloj, empezando con el bit de la fase del reloj puesto a 0 y después a 1.

CPH=0



El flanco de bajada del pin SS seleccionado como 'slave', activa al 'Slave' para empezar la transmisión manejando el pin MISO con el MSB del registro de desplazamiento 'Slave'. El 'Master' empieza la transmisión manejando el MSB del registro de desplazamiento del Master en el pin MOSI. Entonces el 'Master' manejará el reloj SPSCK del SPI. Al primer flanco del reloj SPSCK, ocurre la captura 'strobe'. Esto completa la transferencia de los bits MSB del Slave desde el Master y desplaza los datos en el registro de desplazamiento. Después del siguiente flanco del reloj SPSCK, el 'master' y el 'slave' empezarán manejando los seis bits hacia sus pins respectivos. El ciclo continúa hasta que todos los 8-bits se han transferido.

CPH=1



Cuando el pin $\overline{SS}$ 'slave' pasa a nivel bajo, el 'Slave' espera hasta el primer flanco del reloj SPSCK para manejar su pin MISO. Los bits MSB del 'master' y el 'slave', manejan a sus respectivos pins al primer flanco de reloj SPSCK. Ocurre la captura 'strobe' en el segundo flanco de reloj SPSCK con CPH puesto a 1, en lugar del primer flanco cuando CPH está puesto a 0. Finalmente, el ciclo continúa hasta que todos los 8-bits se han transferido.

Ejercicio de programación del SPI

Se trata de escribir un programa para configurar al SPI master para la transmisión de datos. En este ejercicio, se realiza haciendo una exploración de una transferencia SPI de un byte a un registro de desplazamiento de entrada serie y de salida paralela de 8-bits. Se podría usar este programa para proporcionar puertos adicionales de entrada/salida con 'latch' a través del registro de desplazamiento.

En este programa, hay que configurar el SPI como se maneja una no-interrupción por el master. Transmitir un byte de datos que contienen el valor $55. El registro de desplazamiento se debe seleccionar con la línea PB3 del puerto usando un bajo voltaje seleccionado. Hay que asegurarse que la línea BP3 permanece en estado alto entre las transferencias.

Configurar el reloj SPI para estar en modo 'idle low' y tomar los datos en los flancos de subida del reloj. También, configurar la frecuencia del reloj SPI a 300 kHz y asumir una frecuencia de reloj del sistema de 8 Mhz.

```
PTB     EQU     $01             ; Registro de Datos Puerto B
DDRB    EQU     $05             ; Registro de Dirección de Datos del Puerto B
SPCR    EQU     $10             ; Registro de Control SPI
SPSCR   EQU     $11             ; Registro de Estado y Control SPI
SPDR    EQU     $12             ; Datos (Lectura = rcv, Escritura = xmt)
                                ;
                                ; Selección de entrada/salida y niveles del Puerto B:
                                ;
        BSET    #3,PTB          ; Pone la salida PB3 en estado alto
        BSET    #3,DDRB         ; Pone PB3 como una salida
        LDA     #$06            ; Carga el acumulador (ACC) con:
                                ;     "    MOSI y SPSCK = para outputs
                                ;     "    MISO y SS* = para inputs
        STA     DDRF            ; Guarda el ACC en el registro DDRF
                                ;
                                ; Selecciona modo master, reloj, y activa el SPI:
        LDA     #$02            ; Carga el ACC con el valor que selecciona 300 kHz
        STA     SPSCR           ; Guarda el ACC en el registro SPSCR
        LDA     #$22            ; Carga el ACC con valor a configurar SPSCR
        STA     SPCR            ; Guarda el ACC en el registro SPCR
                                ;
                                ; Envía $55 al registro de desplazamiento:
                                ;
        BCLR    #3,PORTB        ; Pone la salida PB3 en estado bajo (registro chip-select)
        LDA     #$55            ; Carga el ACC con $55
        STA     SPDR            ; Guarda el ACC en SPDR
WAIT    BRCLR   #7,SPSCR,WAIT   ; Espera hasta que la transmisión sea completada
        BSET    #3,PORTB        ; Pone salida PB3 en estado alto (registro de selección)
DONE    BRA     DONE            ; Hecho...
```

Módulo ADC (Analog to Digital Converter)

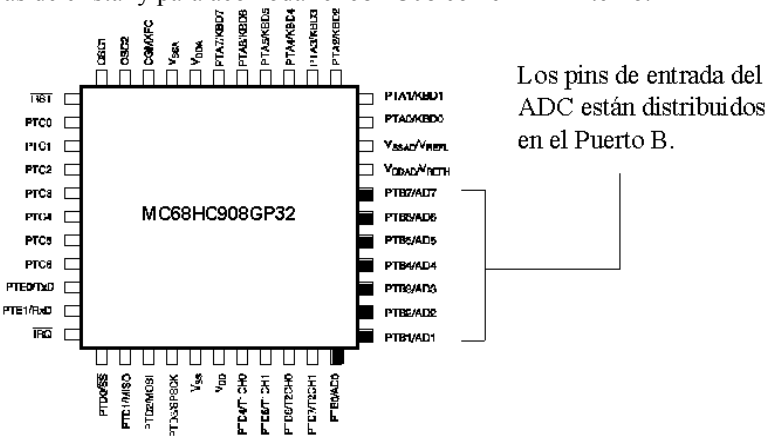
En esta parte se describen las características y la configuración del Módulo ADC. Así como las técnicas para obtener la máxima precisión en las conversiones del ADC y se escribe un programa para hacer medidas con el convertidor.

Características del ADC

Algunas MCU de la familia 68HC08 incluyen un convertidor A/D de 8-bits o 10-bits de resolución. Algunos ejemplos del 68HC08 con un ADC de 8-bits son el 68HC908GP, GR, JL, JK y KX. Algunos ejemplos del 68HC08 con un ADC de 10-bits son el 68HC908MR y 68HC908SR. Todos los Módulos ADC de los 68HC08 usan el principio de aproximaciones sucesivas. Una discusión de este principio se puede encontrar en la el Manual de Referencia del M68HC11 M68HC11RM/AD (véase en http://www.mcu.motsps.com)

Todos los Módulos ADC soportan dos modos de conversión, en modo de conversión continua y en modo de conversión única. En modo de conversión única, se completa una conversión entre escribir el registro de estado y el registro de control. En modo de conversión continua, la entrada analógica del ADC convierte continuamente y lo escribe en el registro de datos del ADC. En este modo, los datos de la conversión anterior se borran sin tener en cuenta si estos datos han sido leídos o no.

El ADC ofrece dos maneras diferentes de supervisar el estado de conversión completa. Dependiendo del modo de conversión, cuando se ha completado una conversión, se puede usar el software para consultar un indicador (‘flag’) o se puede configurar el ADC para generar una señal de interrupción. El ADC tiene un reloj de entrada seleccionable. Se puede usar este reloj de entrada para optimizar las conversiones del ADC para diferentes frecuencias de cristal y para acomodar el 68HC08 con el PLL interno.



Configuración del ADC

Para configurar el Módulo de ADC del 68HC908GP32 para hacer conversiones, primero hay que configurar el pin de alimentación analógica V_{DDAD}. Normalmente se conecta este pin al mismo voltaje que el pin de alimentación digital V_{DD}. Para lograr un buen funcionamiento del ADC, es necesario usar un filtrado externo para asegurar un voltaje limpio en V_{DDAD}. Para la máxima inmunidad al ruido, hay que rutar cuidadosamente V_{DDAD} y poner condensadores de desacoplo tan cerca como sea posible al encapsulado del microcontrolador. El ADC usa el pin V_{SSAD} como pin de tierra analógica. Se debe conectar este pin al mismo potencial que V_{SS}. En esta configuración, el pin de tensión de referencia alto V_{REFH}, está separado del pin de tensión de referencia bajo V_{REFL}. El pin V_{REFH} está compartido con el pin de alimentación V_{DDAD} y V_{REFL} está compartido con el pin de tierra V_{SSAD}.

Una vez se ha configurado el Módulo ADC como se ha descrito, está listo para convertir el voltaje de entrada V_{ADIN} a un valor digital. La señal de entrada se lee desde el canal seleccionado de entrada del ADC en el registro de estado y de control del ADC. El resultado de la conversión depende del valor de V_{ADIN}.

El pin V_{REFH} (tensión de referencia alta) se conecta al pin de alimentación analógica V_{DDAD} del ADC y el pin V_{REFL} (tensión de referencia baja) se conecta al pin de tierra analógica V_{SSAD} del ADC. Por consiguiente, V_{ADIN} no debe exceder la tensión de alimentación analógica.

Si el valor de V_{ADIN} está entre V_{REFH} y V_{REFL}, el ADC convierte el voltaje usando una conversión lineal. El resultado es uno de los 256 valores digitales que van de $00 a $FF. Si V_{ADIN} se iguala a V_{REFH}, el ADC convierte la señal a $FF. Si V_{ADIN} se iguala V_{REFL}, el ADC lo convierte a $00.

El proceso de la conversión es monotónica y no tiene ninguna perdida de código. Esto significa que si se aumenta el voltaje de la entrada, el ADC convierte la señal todos los valores entre $00 y $FF. En este caso, el siguiente resultado de la conversión será siempre más alto que el anterior.

Registros del Módulo ADC

- **(ADSCR) Registro de estado y de control del ADC**
- **(ADR) Registro de datos del ADC**
- **(ADCLK) Registro del reloj del ADC**

Se puede controlar y monitorizar el funcionamiento del ADC usando estos tres registros. Usando el registro ADSCR de estado y de control del ADC, se puede configurar el canal de entrada analógica y el modo de la conversión, y también supervisa el estado de conversión completa. Después de convertir el voltaje de entrada, el ADC escribe el resultado en el registro ADR de datos del ADC. Se puede configurar el reloj de entrada del ADC usando el registro ADCLK del reloj del ADC.

Todos los registros del ADC están en el mapa de memoria. Para el ADC del 68HC908GP32, el ADSCR está en la posición de memoria $003C, el ADR está en la posición de memoria $003D y el ADCLK está en la posición de memoria $003E.

Registro ADSCR de estado y de control

ADSCR

$0003C	Bit 7	6	5	4	3	2	1	0
Leer: Escribir:	COCO/ IDMAS	AIEN	ADCO	ADCH4	ADCH3	ADCH2	ADCH1	ADCH0
Reset:	0	0	0	1	1	1	1	1

• El indicador *COCO*, es un bit de sólo lectura e indica cuando una conversión se ha completado. En modo de una sola conversión, el ADC pone el indicador COCO a 1, después de que cada conversión hay sido completada. En modo de conversión continua, el ADC pone el indicador COCO a 1, después de que la primera conversión haya sido completada.

• En el 68HC08 que tiene el módulo de acceso directo a memoria (DMA), se puede usar el bit 7 para controlar el funcionamiento del DMA. En este tipo de configuración, el bit 7 ADSCR se refiere a IDMAS. Sólo se debe escribir en esta posición si el dispositivo tiene DMA. Escribiendo en esta posición, en las MCU que no contienen DMA, se enmascararán las interrupciones del ADC y causará resultados no deseados.

• Se pone el indicador *AIEN* a 1 para configurar el ADC para generar una señal de interrupción cuando la conversión se haya completado. La señal que genera la interrupción se pone a 0, cuando el registro de datos se lee o se escribe el registro de estado y de control.

• Se puede seleccionar el modo de conversión con el indicador *ADCO* de conversión continua. Poniendo este bit a 1 se selecciona el modo de conversión continua. Cuando este bit se pone a 0, el ADC se configura en modo de conversión simple.

• Los 5 bits restantes del ADSCR, *ADCH4–ADCH3–ADCH2–ADCH1–ADCH0* del ADSCR, contienen los bits de selección de los canales del ADC. En el ADC del 68HC908GP32, se pueden usar estos de bits para seleccionar uno de los ocho canales de entrada (AD0 - AD7) y verificar el funcionamiento del ADC.

Selección del Canal de Entrada

La siguiente tabla muestra los códigos de selección de canales del ADC del 68HC908GP32. Con estos bits también se puede seleccionar V_{REFH} y $V_{SSAD,}$ así como la señal de entrada para verificar el funcionamiento del ADC.

ADCH4	ADCH3	ADCH3	ADCH1	ADCH0	Selección de entrada
0	0	0	0	0	**PTB0/AD0**
0	0	0	0	1	**PTB1/AD1**
0	0	0	1	0	**PTB2/AD2**
0	0	0	1	1	**PTB3/AD3**
0	0	1	0	0	**PTB4/AD4**
0	0	1	0	1	**PTB5/AD5**
0	0	1	1	0	**PTB6/AD6**
0	0	1	1	1	**PTB7/AD7**
↓	↓	↓	↓	↓	**Reservado**
1	1	0	1	1	**Reservado**
1	1	1	0	0	**Reservado**
1	1	1	0	1	**V_{REFH}**
1	1	1	1	0	**V_{SSAD}**
1	1	1	1	1	**Desconexión del ADC**

Cuando no se necesita el ADC, se puede desconectar poniendo todos los canales seleccionados del ADC a 1. Esto ayudará a minimizar el consumo del sistema. Si se selecciona un canal de entrada sin usar o un código de bit reservado, el resultado de la conversión del ADC será desconocido.

Registro ADR de datos del ADC

El registro de datos ADR del ADC es un registro de sólo lectura que el ADC usa para guardar los resultados de la conversión. El ADC actualiza el registro ADR después de que haya sido completada cada conversión.

ADR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
Escribir:								
Reset:	0	0	0	0	0	0	0	0

Registro ADCLK del reloj del ADC

Usando los bits del prescaler del reloj del ADC, ADIV2 – ADIV1 – ADIV0, se puede seleccionar uno de los cinco valores del divisor: 1, 2, 4, 8 o 16. El ADC genera la frecuencia de reloj dividiendo la fuente de reloj por el valor del divisor seleccionado.

ADCLK	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	ADIV2	ADIV1	ADIV0	ADICLK	0	0	0	0
Escribir								
Reset:	0	0	0	0	0	0	0	0

Se puede seleccionar la fuente del reloj de entrada del ADC, usando el bit ADICLK de selección de reloj del ADC. Si este bit se pone a 0, el ADC usará el reloj externo CGMXCLK como reloj de entrada. Si se pone a 1, el ADC usará el reloj del PLL interno del 68HC08 como reloj de entrada.

Se debe seleccionar la fuente de reloj de entrada basándose en la frecuencia del reloj. El módulo del ADC se ha diseñado para operar mejor con una frecuencia de reloj de entrada de 1 MHz. Si la frecuencia de reloj externa es mayor o igual a 1 MHz, se debe usar el reloj externo como fuente de entrada. Por otra parte, hay que usar el reloj interno del bus como fuente de entrada.

Cálculo del Tiempo de una Conversión

Una vez se ha seleccionado la fuente de reloj de entrada, se puede calcular la cantidad de tiempo que se usa para completar una sola conversión. Primero se determina el número de ciclos de reloj que se usa para completar la conversión y entonces dividir este valor por la frecuencia de reloj de entrada.

$$\textbf{Tiempo de una Conversión} = \frac{\textbf{16 o 17 (ciclos del ADC)}}{\textbf{1MHz (Frecuencia del ADC)}} = \textbf{16 o 17}\mu\textbf{s}$$

El proceso de conversión empieza después de que se ha escrito el registro ADSCR. Para completar una conversión típica usa 16 ciclos de reloj del ADC. Cuando hay un retraso de sincronización del reloj, entre el reloj de la CPU y el reloj del ADC, para completar la conversión usará 17 ciclos de reloj. Es posible un retraso de sincronización del reloj si el reloj del A/D es diferente que el reloj de la CPU. Con la frecuencia de reloj de entrada puesta a 1MHz, una conversión típica usará aproximadamente de 16 a 17µs.

Mejora de la Precisión de la Conversión

Se verán algunas técnicas que se pueden usar para mejorar la precisión de la conversión A/D. El método más eficaz para mejorar la precisión, es reducir el ruido que se introduce en el subsistema del A/D, usando un trazado del circuito impreso muy estudiado. Donde sea posible, separar las señales con ruido, de las señales sensibles al A/D. Si la separación de estas señales no se puede hacer, hay que reducir el ruido acoplado tanto como sea posible. Para más información sobre la reducción del ruido, véase la nota de aplicación AN1059/D System Design and Layout Techniques for Noise Reduction in MCU Based Systems en la web http://www.mcu.motsps.com.

Después de alimentar el subsistema del A/D, hay que dejar un tiempo para que se estabilice el A/D, antes de empezar una conversión. Este tiempo se proporciona en el libro de datos técnico. También se debe verificar que la impedancia de la fuente de entrada no sea demasiado grande. Los errores causados por la corriente de fuga de entrada del A/D aumenta en proporción a la fuente de impedancia. Si el ancho de banda lo permite, se puede reducir el impacto del ruido inyectado, tomando múltiples conversiones y promediando los resultados.

Ejercicio de Programación del ADC

Ahora se puede escribir un programa para configurar el Módulo ADC del 68HC908GP32. El programa principal debe leer la entrada de un canal del ADC y mostrar los resultados en un conjunto de 8 LEDs. Después de completar una medida, el programa debe repetir el proceso.

Se inicializa el programa con la configuración del ADCLK con un reloj de 1MHz, asumiendo un reloj del sistema de 8 MHz. Se usa el puerto D para manejar los LEDs y el canal 0 del ADC como entrada. Hay que asegurarse de que se desconecta el Módulo COP y el Módulo LVI.

Se usa una subrutina para realizar la medida del ADC. El programa principal debe usar el acumulador para pasar el canal de entrada deseado del ADC a la subrutina. En la subrutina, hay que verificar que un bit del puerto DDR se ha borrado para el canal de entrada del ADC. La subrutina debe devolver el resultado de la conversión al acumulador.

El programa principal inicializa el Módulo ADC del 68HC908GP32 fuera de 'reset' y llama a una subrutina para tomar medidas del A/D. En reset, el primer paso es preparar el registro de configuración del hardware. Es mejor hacer esto explícitamente, aun cuando el estado predefinido, es lo que se quiere.

Las declaraciones de las directivas del ensamblador no se requieren para correr el programa propiamente y se usan para hacer más legible el código fuente. También hacen más fácil a la hora de traducir el programa para correr en diferentes microcontroladores 68HC08. Por ejemplo, si el ADSCR se encuentra en una posición de memoria diferente en una MCU diferente, sólo se necesita cambiar la declaración de la directiva ADSCR, para hacer que el programa corra en otro microcontrolador 68HC08. Si la posición real, en hexadecimal del ADSCR se hubiera usado donde se necesitaba por el código, el programa tendría que ser revisado en múltiples lugares. En un programa de aplicación real, las declaraciones típicamente se ponen en un archivo separado y se incorporaran en el código usando una declaración "include".

En este chip el programa corre fuera de la memoria Flash, por lo que se necesita decirle al compilador donde se localiza en la memoria. La directiva "org" hace esto. Cuando el programa se compila en un fichero S-record, la primera línea de ese registro contendrá esta información y el programador de la Flash lo usará para poner el programa en el lugar correcto.

No se usa ninguna variable en la RAM en este simple programa, por lo que el próximo paso es inicializar las variables que sólo tienen que ser puestas una vez. El Puerto B0 siempre será el puerto de entrada, el Puerto D siempre será el puerto de salida y el reloj del ADC siempre correrá a la misma frecuencia. Por consiguiente, se ponen estas variables antes del lazo del programa principal.

```
ADSCR   equ     $003C                   ;registro de estado y de control del A/D
COCO.   equ     7                       ;Indicador de conversión completada en ADSCR
ADR     equ     $003D                   ;registro de datos del A/D
ADCLK   equ     $003E                   ;registro del reloj del A/D
ADIV0.  equ     5                       ;reloj de entrada ÷ 2 bit
ADIV1.  equ     6                       ;reloj de entrada ÷ 4 bit
        org     FLASH
Reset   mov     #FF, CONFIG1            ;Pone el puerto D como salida para los LEDs
                                        ;Los LEDs mostrarán los resultados
                                        ;del ADC en binario.
        mov     #%01100000,ADCLK        ;El PCB usa un cristal oscilador de 8MHz
                                        ;divide la entrada de reloj del ADC por 8
                                        ;para dar un reloj de 1MHz, recomendado.
        bclr    0,DDRB                  ;Pone PTB0 a la entrada del ADC
main    clra                            ;Se escoge ADC0, pero se podrían usar los canales 0-7
        jsr     adc_meas                ;toma la medida en ADC0
        coma                            ;Los LEDs son lógica negativa en el PCB
        sta       PTD                   ;muestra el resultado
        bra       main                  ;empieza en el lazo principal otra vez
```

El lazo del programa principal llama a una subrutina para tomar una medida y muestra la medición en los LEDs. La subrutina maneja todas las actividades relacionadas con el ADC. La subrutina necesita activar el reloj. También necesita un pin del puerto y un canal de entrada. El reloj del ADC y el puerto están preparados en reset y se necesita decir a la subrutina qué canal se usa, esto se hace cargando la elección del canal en el acumulador.

Cuando vuelve de la subrutina, los 8-bits de la medida están en el acumulador. El circuito se monta para que cada LED se ilumine cuando su pin en el puerto D se pone a un nivel bajo. Una lectura de todo a cero, encendería todos los LEDs. Este valor puede ser fijado para cumplimentar el valor del acumulador antes de escribirlo fuera del puerto. Todos los LEDs están apagados cuando es una medición de señal cero y están todos encendidos cuando se lee a máxima escala. Se guarda el valor usando el puerto D y el lazo vuelve al principio.

```
*************************************************************************
* NOMBRE DE SUBRUTINA:  adc_meas     FECHA DE REVISION: 01/02/2000       *
*                                                                        *
* PROPOSITO:  realizar una conversión A/D en un canal del ADC            *
*                                                                        *
* CONDICIONES DE ENTRADA: El registro ADCLK pone el reloj del ADC a 1MHz *
*                     Canal de entrada deseado del ADC en el acumulador  *
*                     Un bit del Puerto DDR borrado para este canal      *
*                                                                        *
* CONDICIONES DE SALIDA:    Resultado de la Conversión en el acumulador  *
*************************************************************************
adc_meas ora   #%00100000            ;Selecciona el modo de conversión continua
                                     ; con el canal seleccionado
sta     ADSCR                        ;Enciende el ADC y espera para una
        brclr     COCO.,ADSCR,*      ;conversión completa y para que se estabilice
lda     ADR                          ;Borra el bit COCO leyendo ADR
        brclr     COCO.,ADSCR,*      ;este tiempo toma la medida real
        mov       #%00011111,ADSCR   ;Después se apaga el ADC
        lda       ADR                ;Recupera el resultado del ADC
        rts                          ;vuelve con el resultado en el acumulador
```

En este ejercicio, la medida del ADC es tan simple que se podría usar el lazo del programa principal fácilmente al tomar la medida. Sin embargo, la subrutina es un buen ejemplo de reusabilidad del código. Se puede llamar en cualquier parte de un programa más grande. Por ejemplo, suponiendo que se tiene una aplicación que registra ocho sensores de temperatura diferentes en cada uno de los pins del puerto B. Esta rutina se puede usar para todos ellos. La única entrada que se necesita es el identificador del canal de entrada del ADC.

Al ADC se le necesita decir si se va a usar en modo de conversión simple o conversión continua. Esto se hace seleccionando el bit 5 del ADSCR. Luego, combina el bit de conversión seleccionado con la opción de canal de entrada en un valor de 8-bits. La mejor manera de hacer esto es con la operación lógica OR. Por ejemplo, seleccionando el modo de conversión continua usando la entrada del canal 3. Se usa el acumulador para pasar a la subrutina un “3” para el canal de entrada. Si se hace una operación lógica "OR" del acumulador con $20, para el modo de conversión continua, el resultado es $23.

En cuanto se guarde este valor en ADSCR, el Módulo ADC empieza a convertir. El circuito del ADC no se ha estabilizado todavía, por lo que se necesita esperar un ciclo de conversión para estabilizarse. La manera más directa de resolverlo, es hacer un lazo continuo mientras se registra el bit de conversión completa en el ADSCR. Cuando este bit está a 1, la conversión se ha completado y el circuito ha tenido tiempo para estabilizarse. Este paso no se requiere, si se empieza convirtiendo después de que el Módulo ADC se haya alimentado.

Leyendo el ADR, se borra el bit COCO. No se usará la primera medida que fue tomada cuando el circuito del ADC estaba inestable. En cambio, se espera para completar la próxima medida. En cuanto esté listo, el bit COCO se pone de nuevo a 1 y se saldrá del lazo. Se apaga el Módulo ADC poniendo el bit de selección de canal del ADC en el ADSCR. Entonces se lee la medida desde el ADR y devuelve el resultado en el acumulador.

Mejoras: Ahora se van a considerar algunas mejoras que se pueden hacer en esta solución para aplicaciones del mundo real.

Antes se ha descrito de que se puede usar un algoritmo de promedio para filtrar los datos. Hay muchas maneras de lograr esto, dependiendo de cómo se quiere filtrar. Una opción que siempre se debe considerar, es utilizar un filtro hardware en la entrada. Si se sabe que la señal real que se está intentando medir no puede cambiar significativamente en menos de un segundo, el punto de cambio más cercano estará en el milisegundo. Se puede poner un filtro pasa-bajo en el canal de entrada, para bloquear el ruido.

Si se quiere usar el "stack" en lugar del acumulador para pasar argumentos a la subrutina, si hay sólo un argumento y el acumulador está disponible, no hay nada malo con este método.

El método lazo y el método consulta ("poll") en esta subrutina pueden causar un lazo infinito. Si por alguna razón el bit COCO nunca se pone a 1, el código se mantendrá en el lazo y nunca se recuperará. Otro problema con este método es que la CPU pierde tiempo. Un método alternativo es configurar el ADC para activar un evento de interrupción cuando el bit COCO se pone a 1. Usando este método, el programa puede hacer otras cosas útiles mientras se espera que el ADC termine su conversión.

Con un programa de manejo de interrupción, se puede poner la cantidad de tiempo que el programa espera para completar una conversión del ADC. Si el ADC ha estado convirtiendo durante 20 ciclos de reloj y el bit COCO todavía no está a 1, el programa podría bifurcar para realizar una rutina de error e informar que algo está mal. Este tipo de programación de tolerancia de fallo está más allá del alcance de este ejercicio, pero forma parte importante de una estrategia global para producir software robusto en las aplicaciones.

Módulo Generador de Reloj

En este Módulo Generador de Reloj (CGMC) del 68HC08, se describen las características y su configuración. Se selecciona una fuente de reloj base apropiada, se calculan los diferentes valores requeridos para programar y se configura el PLL para generar una frecuencia de bus específica.

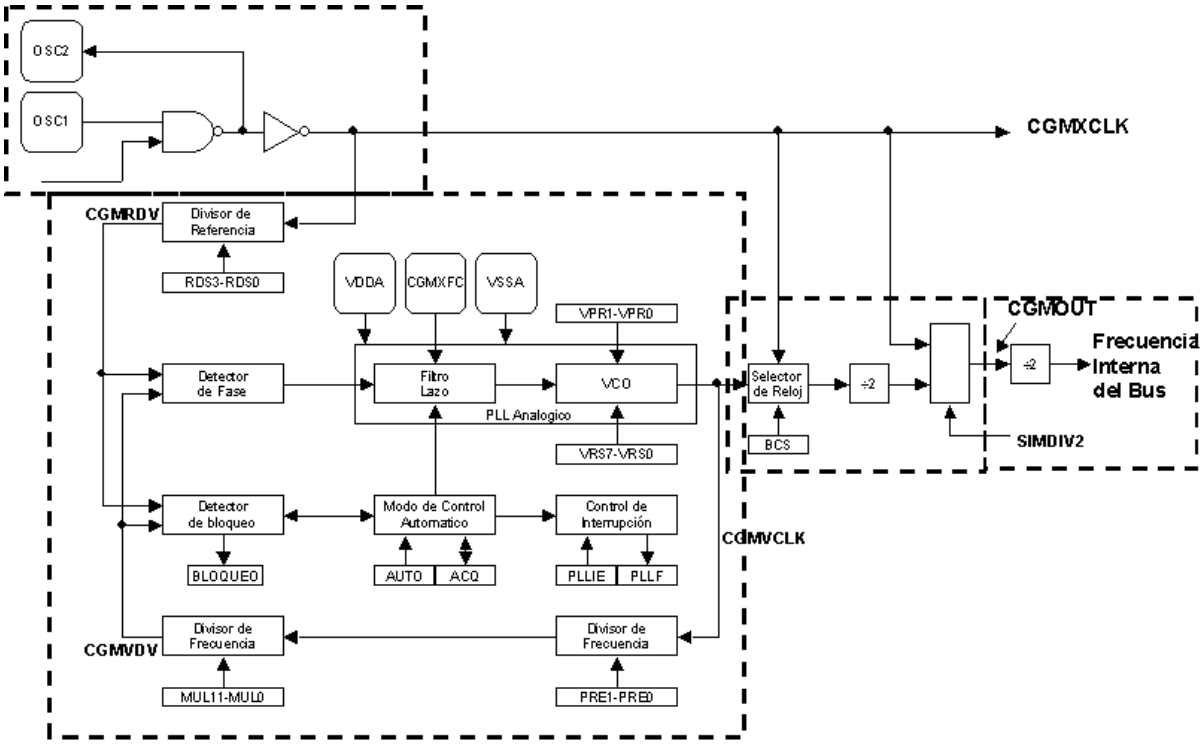
El CGMC incluye un PLL integrado que es capaz de generar frecuencias en múltiplos enteros del cristal de referencia. Por ejemplo, usando el PLL se puede proporcionar hasta una frecuencia de bus de 8.2Mhz usando un cristal de 32 Khz, de muy bajo costo. El uso del PLL reduce la generación de ruido, en comparación con los cristales de frecuencias más altas y no son necesarios los osciladores.

El módulo CGMC tiene dos prescalers programables, que aumentan por dos la frecuencia. También incluye un hardware con un oscilador programable por tensión (VCO) para un funcionamiento con bajas fluctuaciones (jitter). El Módulo CGMC puede cambiar automáticamente del modo "adquisición" al modo "rastreo" (tracking), dependiendo de cuanto esté desplazada la frecuencia de salida con respecto a la frecuencia de referencia.

El PLL detecta automáticamente cuando la frecuencia de salida está bastante cerca a la frecuencia deseada y se bloquea (permanecerá en esa frecuencia). La CPU se puede configurar para generar una interrupción cuando la frecuencia deseada se bloquea o sale de la condición de bloqueo. En el registro CONFIG, se puede poner a 1 el bit OSCSTOPENB para permitir que el oscilador opere durante el modo STOP.

Diagrama de Bloques del CGMC

El CGMC consiste en tres submódulos: el circuito del oscilador a cristal, el PLL y el circuito de selección de reloj base.



El circuito del oscilador a cristal consiste en un amplificador inversor y un cristal externo. El pin OSC1 es la entrada del amplificador inversor y el pin OSC2 es la salida. El circuito del oscilador a cristal genera la frecuencia de reloj del cristal constante, CGMXCLK corre a una velocidad igual a la frecuencia del cristal. CGMXCLK pasa por un "buffer" para producir la referencia de reloj del PLL.

El PLL es un generador de frecuencia totalmente funcional, que ha estado diseñado para su uso con cristales o con resonadores cerámicos. El PLL genera la frecuencia de reloj programable CGMVCLK con el VCO,. La frecuencia del VCO será un múltiple entero de la frecuencia de referencia. El circuito de selección del reloj base CGMOUT se controla por software. Se puede poner el reloj base a CGMXCLK dividido por dos o CGMVCLK dividido por dos.

El Módulo SIM (Módulo Integración del Sistema) posteriormente deriva los relojes del sistema desde CGMOUT o CGMXCLK, con una división adicional a través de dos circuitos, para producir la frecuencia interna del bus. Esto se verá más adelante.

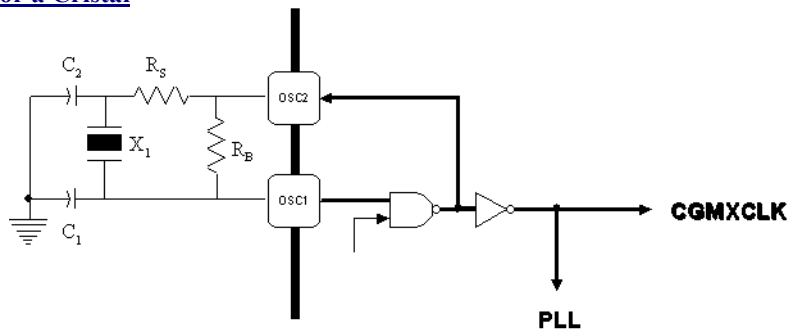
Características del Oscilador a Cristal

El CGMXCLK puede ser usado por módulos que requieren trabajar con unos tiempos muy precisos. El ciclo de servicio de CGMXCLK no se garantiza que sea del 50% y depende de factores externos, incluyendo el cristal y otros componentes externos. También se puede usar un reloj generado externamente entrando por el pin OSC1 del circuito oscilador a cristal. Para usar esta configuración, hay que conectar el reloj externo al pin OSC1 y dejar el pin OSC2 al aire.

El circuito oscilador a cristal puede tomar dos fuentes de frecuencia diferentes, la de un cristal o la de un oscilador externo. Si se elige usar un cristal, la frecuencia del cristal debe estar entre 30 kHz y 100 kHz, típicamente de 32.768 kHz. El oscilador no se ha diseñado para frecuencias de cristal fuera de este rango.

Si se elige usar un oscilador externo como fuente de reloj, se puede elegir tener el PLL activado o se puede desactivado. Con el PLL activado, la frecuencia del oscilador externo debe estar entre 30 kHz y 1.5 MHz. Con el PLL desactivado, la frecuencia del oscilador externo puede ir de cero a 32 MHz.

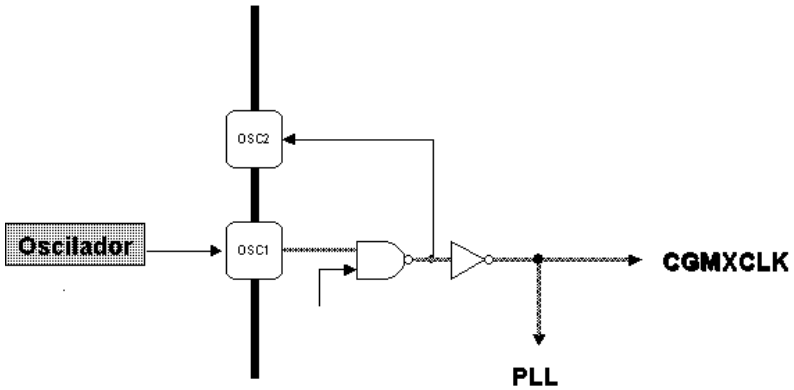
Circuito Oscilador a Cristal



Al usar un cristal como fuente de reloj, la frecuencia se puede seleccionar como un reloj base o como un reloj de referencia para el PLL. En esta configuración, el PLL se puede activar o se puede desactivar. La frecuencia del cristal debe estar entre 30 kHz y 100 kHz, para garantizar un buen funcionamiento.

Oscilador Externo

Con la configuración de oscilador externo, se puede trabajar con el PLL activado o desactivado dependiendo de la frecuencia del oscilador.



- Con el PLL activado, se puede usar un oscilador externo con una frecuencia entre 30Khz y 1.5 Mhz.
- Con el PLL desactivado, se puede usar un oscilador externo con una frecuencia entre DC y 32.8 MHz.

Características del PLL

Como se vio antes, el PLL genera la frecuencia del VCO programable. El PLL puede filtrar la frecuencia del VCO, usando dos modos: modo adquisición y modo rastreo (‘tracking’). La selección del modo, depende la exactitud de la frecuencia de salida.

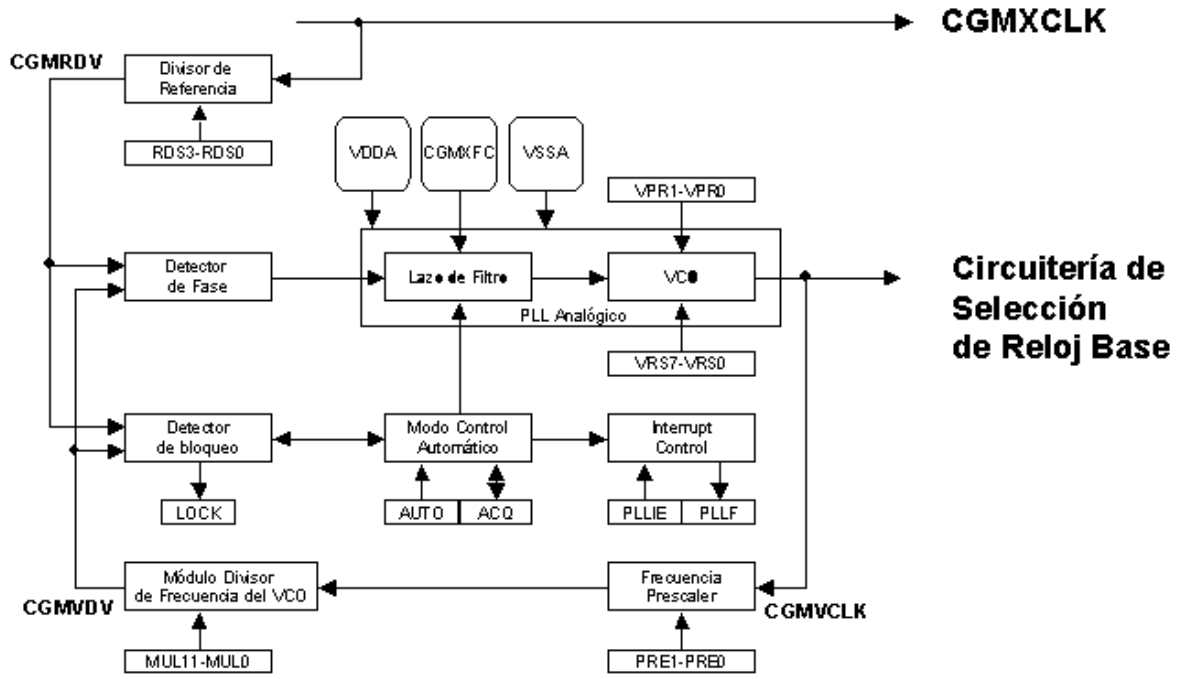
En *modo adquisición*, el filtro puede hacer correcciones de frecuencia grandes a la frecuencia del VCO. Este modo se usa al iniciar del PLL o cuando el PLL ha sufrido un ruido severo y la frecuencia del VCO está desplazada de la frecuencia deseada. Para seleccionar este modo, hay que seleccionar el bit ACQ en el registro de control del ancho de banda del PLL.

En *modo rastreo ('tracking')*, el filtro hace pequeñas correcciones a la frecuencia del VCO. Las fluctuaciones ('jitter') del PLL son muy bajas en modo "tracking", pero la respuesta al ruido también es más lenta. El PLL entra en modo 'tracking' cuando la frecuencia del VCO es casi correcta, como cuando el PLL se selecciona como fuente de reloj base.

El PLL puede cambiar entre el modo adquisición y modo rastreo, automáticamente o manualmente. Se recomienda el modo automático para la mayoría de las aplicaciones. Hay muchas ventajas al incluir un PLL en la MCU. Se puede lograr una frecuencia de bus alta con un cristal económico, ya que los cristales de baja frecuencia son más baratos. Los cristales de baja frecuencia también consumen menos y proporcionan más alta inmunidad al ruido.

Circuito PLL

El PLL es básicamente un oscilador cuya frecuencia se bloquea en una componente de frecuencia de una señal de entrada. En este ejemplo, el PLL actúa como un multiplicador de frecuencia para lograr una frecuencia de alta resolución de salida, usando un cristal de baja frecuencia se minimiza el costo, consume menos y genera menos ruido.



CGMRCLK es el reloj de referencia del PLL, que pasa a través de un "buffer", siendo entonces CGMXCLK. CGMRCLK corre a una frecuencia f_{RCLK} y alimenta al PLL a través de un módulo programable divisor de referencia, que divide la frecuencia por un factor R. La salida del divisor, es el reloj de referencia final, CGMRDV.

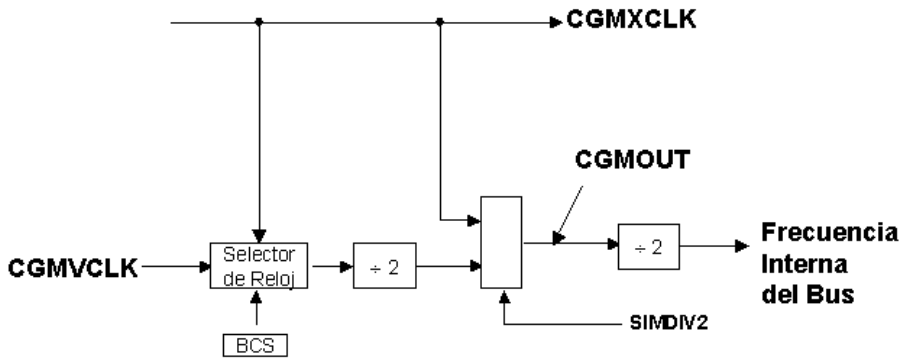
La salida de reloj del VCO, corre a una frecuencia, f_{VCLK} y se realimenta a través de un divisor programable "prescaler" y a través de un divisor programable "VCO". El divisor "prescaler" divide el reloj del VCO por un factor potencia de dos (P) y el módulo divisor de frecuencia VCO reduce esta frecuencia por un factor N. La salida es la de realimentación del reloj del VCO, CGMVDV, que corre a una frecuencia f_{VDV}.

El detector de fase compara CGMVDV con CGMRDV y genera un pulso de corrección basado en la diferencia de la fase de las dos señales. El filtro del lazo altera ligeramente el voltaje DC en el condensador externo conectado a CGM/XFC basado en la anchura y en la dirección del pulso de corrección.

El detector de bloqueo también compara CGMVDV con CGMRDV. La frecuencia se bloquea cuando la diferencia de la fase entre las dos señales está muy cerca de cero.

Circuito Selector de Reloj Base, SIM Divisor de Reloj

Ahora se puede ver el último sub-módulo CGMC, el circuito de selección de reloj base. Este circuito se usa para seleccionar CGMXCLK o CGMVCLK como fuente de reloj base, CGMOUT.



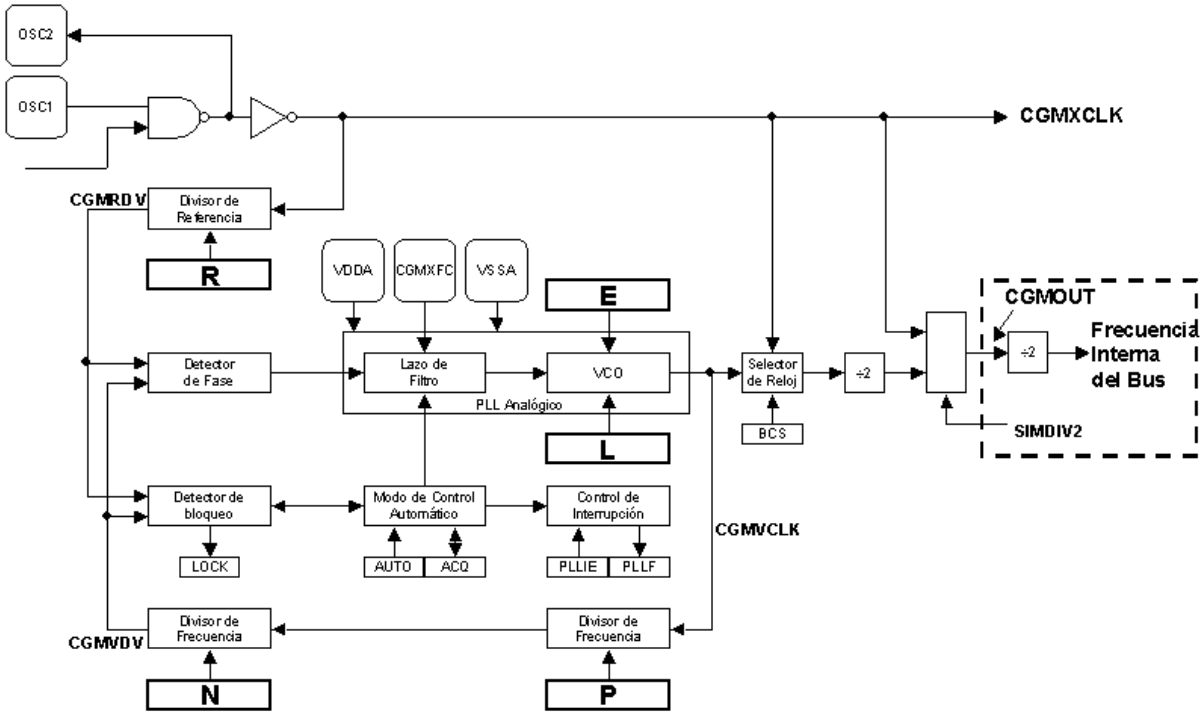
Los dos relojes de entrada pasan por un circuito de control de transición, que espera hasta tres ciclos CGMXCLK y tres ciclos CGMVCLK para cambiar de una fuente de reloj a otra. Durante este tiempo, CGMOUT se mantiene en espera del evento. El bit BCS del registro de control del PLL selecciona qué reloj va a manejar CGMOUT.

La salida del circuito de control de transición, se divide por dos para corregir el ciclo de servicio y al SIM para una división por dos adicional, para generar la frecuencia interna del bus, que es un cuarto de la frecuencia del reloj seleccionado BCS (CGMXCLK o CGMVCLK).

Un modo monitor de funcionamiento especial, descrito en el Módulo Flash, selecciona CGMXCLK directamente a través de la señal SIMDIV2, para proporcionar una frecuencia interna del bus de CGMXCLK dividida por dos. Esta opción no permite usar el PLL. En modo usuario normal, el CGMXCLK se puede seleccionar usando sólo el bit BCS que produce CGMXCLK dividido por 4.

Módulo Generador de Reloj

El CGMC usa varios factores para programar el PLL. **R** es el valor para el divisor de referencia. **E** es el valor del VCO, con un rango multiplicador potencia de dos. **L** es el multiplicador con rango lineal del VCO. **P** es el divisor de frecuencia para el prescaler y **N** es el valor del módulo divisor de frecuencia del VCO.



Programación del PLL

Ahora, se pueden ver los pasos para programar el PLL:

1. Se selecciona la frecuencia del bus, $\mathbf{f_{BUSDES}}$. Ésta es la frecuencia interna a la que opera la MCU.
2. Se calcula la frecuencia $\mathbf{f_{VCLKDES}}$ del VCO deseada, que es cuatro veces la frecuencia del bus, $f_{VCLKDES} = 4 \times f_{BUSDES}$.

3. Se selecciona una frecuencia de referencia **R** del PLL a cristal o a oscilador, f_{RCLK}, y el divisor de referencia del reloj, donde $1 \leq R \leq 15$.

   Los errores de frecuencia al PLL son corregidos a una velocidad de $\frac{f_{RCLK}}{R}$

   También, el VCO puede ser un múltiplo de esta velocidad, donde N y P son enteros.

$$f_{VCLK} = \frac{2^P N}{R} f_{RCLK}$$

   Con un cristal externo entre 30 kHz - 100 kHz, **R** siempre se pone a 1. Con una fuente de reloj externa de frecuencia alta entre 100 kHz - 1.5 MHz, se usa **R** para dividir la frecuencia externa por un valor entre 30 kHz y 100 kHz. Hay que intentar escoger **R** tan bajo como sea posible, ya que los errores de frecuencia en el PLL se corrigen a una velocidad de f_{RCLK} dividido por R y debe ser un número entero. Para mayor estabilidad y reducción de tiempo de bloqueo, esta velocidad debe ser tan rápida como posible.

4. Se selecciona **N** del módulo divisor de frecuencia del VCO, se multiplica **R** por la frecuencia del VCO, y se divide el producto por la frecuencia de referencia y se redondea el resultado al entero más cercano, siendo $1 \leq N \leq 4095$

$$N = \text{redondeo}\left[\frac{R \times f_{VCLKDES}}{f_{RCLK}}\right]$$

5. Se selecciona la frecuencia P del prescaler, basada en un valor de N, donde $0 \leq P \leq 3$.
   Si **N** $\leq N_{max}$ (4095) se usa **P** = 0.
   Si **N** $> N_{max}$, (4095) se usa la tabla siguiente para seleccionar P y se recalcula N con la nueva formula que se muestra:

Valor de N	P
$0 < \mathbf{N} \leq N_{max}$	0
$N_{max} < \mathbf{N} \leq 2N_{max}$	1
$2N_{max} < \mathbf{N} \leq 4N_{max}$	2
$4N_{max} < \mathbf{N} \leq 8N_{max}$	3

$$N = \text{redondeo}\left[\frac{R \times f_{VCLKDES}}{2^P \times f_{RCLK}}\right]$$

6. Se calcula f_{VCLK} y f_{BUS}.

$$f_{VCLK} = \frac{2^P N}{R} f_{RCLK}\ , \qquad f_{BUS} = \frac{f_{VCLK}}{4}$$

7. Se selecciona el rango multiplicador E potencia de dos del VCO, usando la tabla siguiente, donde $0 \leq E \leq 2$:

Rango de Frecuencia	E
$0 < f_{VCLK} <$	0
$9{,}830{,}400 \leq f_{VCLK} < 19{,}660{,}800$	1
$19{,}660{,}800 \leq f_{VCLK} < 39{,}321{,}600$	2

8. Se selecciona un rango multiplicador lineal L del VCO**,** donde f_{NOM} se seleccionó por diseño a 38.4 kHz y $1 \leq L \leq 255$.

$$L = \text{redondeo}\left[\frac{f_{VCLK}}{2^E \times f_{NOM}}\right]$$

9. Se calcula el centro del rango de la frecuencia programada del VCO, f_{VRS}.

$$f_{VRS} = (L \times 2^E) f_{NOM}$$

   Este es el punto medio entre la frecuencia mínima y máxima del PLL, 38.4 kHz $\leq f_{VRS} \leq$ 40 MHz. Para un funcionamiento apropiado,

$$\left|f_{VRS} - f_{VCLK}\right| \leq \frac{2^E \times f_{NOM}}{2}$$

10. Se verifica la opción de P, R, N, E y L por comparación f_{VCLK} con f_{VRS} y $f_{VCLKDES}$. Para un funcionamiento apropiado, f_{VCLK} puede estar dentro de la tolerancia de la aplicación de $f_{VCLKDES}$ y f_{VRS} debe estar los más cerca posible a f_{VCLK}.
11. Se programan los registros del PLL de acuerdo con:
    - •Se selecciona P usando el bit PRE en el registro de control del PLL (PCTL).
    - •Se selecciona E usando el bit VPR en el registro de control del PLL (PCTL).
    - •Se selecciona N usando el multiplicador seleccionado en la parte baja del registro del PLL (PMSL) y el multiplicador seleccionando en la parte alta del registro del PLL (PMSH).
    - •Se selecciona L usando el rango seleccionado del registro VCO del PLL (PMRS).
    - •Se selecciona R usando el divisor de referencia del registro seleccionado del PLL (PMDS).

Ejemplo de Cálculo de programación del PLL

Ahora, se puede ver un ejemplo específico para demostrar lo simple que es este proceso, haciendo los cálculos para programar el PLL, para generar una frecuencia de bus de 8 MHz con un cristal de 32.768 kHz. Puesto que la frecuencia de referencia está entre 30 kHz y 100 kHz, entonces el divisor de referencia se debe poner a 1.

1. La frecuencia de bus deseada es $f_{BUSDES} = 8$ MHz
2. La frecuencia del VCO deseada es $f_{VCLKDES} = 4 \times f_{BUSDES} = 32$ MHz
3. Se escoge la frecuencia de referencia, $f_{RCLK} = 32.768$ kHz. Puesto que $30\text{ kHz} \le f_{RCLK} \le 100\text{ kHz}$, pone R = 1.
4. Usando los valores anteriores, se calcula el multiplicador N de la frecuencia del VCO:
$$N = \frac{R \times f_{VCLKDES}}{f_{RCLK}} = \frac{1 \times 32\text{MHz}}{32.768\text{ kHz}} = 976.56$$, redondeando **N = 977**
5. Como que el valor N es menor que el valor máximo 4095, se pone el valor del prescaler P = 0, usando la tabla del paso 5.
6. Seguidamente, se verifican los valores de f_{VCLK} y f_{BUS} comparándolos respectivamente con los valores deseados de 32 MHz y 8 MHz.
$$f_{VCLK} = \frac{2^P N}{R} f_{RCLK} = \frac{2^0 \times 977}{1} \times 32.768\text{kHz} = 32.014\text{MHz}$$
$$f_{BUS} = \frac{f_{VCLK}}{4} = \frac{32.014\text{MHz}}{4} = 8.003\text{MHz}$$
Los cálculos producen una frecuencia del VCO de 32.014 MHz y la frecuencia del bus de 8.003 MHz. Estos valores están muy cerca de lo que se deseaba. Las pequeñas diferencias fueron introducidas cuando se redondeó el valor de N para conseguir un valor entero.
7. Se selecciona el rango multiplicador E potencia de dos del VCO, usando la tabla del paso 7.
8. Se calcula el rango multiplicador lineal L del VCO,
$$L = \frac{f_{VCLK}}{2^E \times f_{NOM}} = \frac{32.014\text{ MHz}}{2^2 \times 38.4\text{kHz}} = 208.43$$ redondeando, L = 208
9. f_{VRS}, está dentro del rango apropiado,
$$f_{VRS} = (L \times 2^E) f_{NOM} = 208 \times 2^2 \times 38.4\text{kHz} = 31.9488\text{Mhz}$$
$$38.4\text{ kHz} \le f_{VRS} \le 40\text{ MHz}.$$
Y también satisface la segunda ecuación
$$\left| f_{VRS} - f_{VCLK} \right| \le \frac{2^E \times f_{NOM}}{2}$$ $65200 \le 76800$
10. f_{VRS} tiene que estar los más cerca posible de f_{VCLK},
$$f_{VRS} = 31.9488\text{ MHz} \approx f_{VCLK} = 32.014\text{ MHz}$$
11. Se configuran los registros usando estos valores:

R = 1	Registro PMDS
N = 0x03D1	Registros PMSL y PMSH
P = 0	Registro PCTL
E = 2	Registro PCTL
L = 0xD0	Registro PMRS

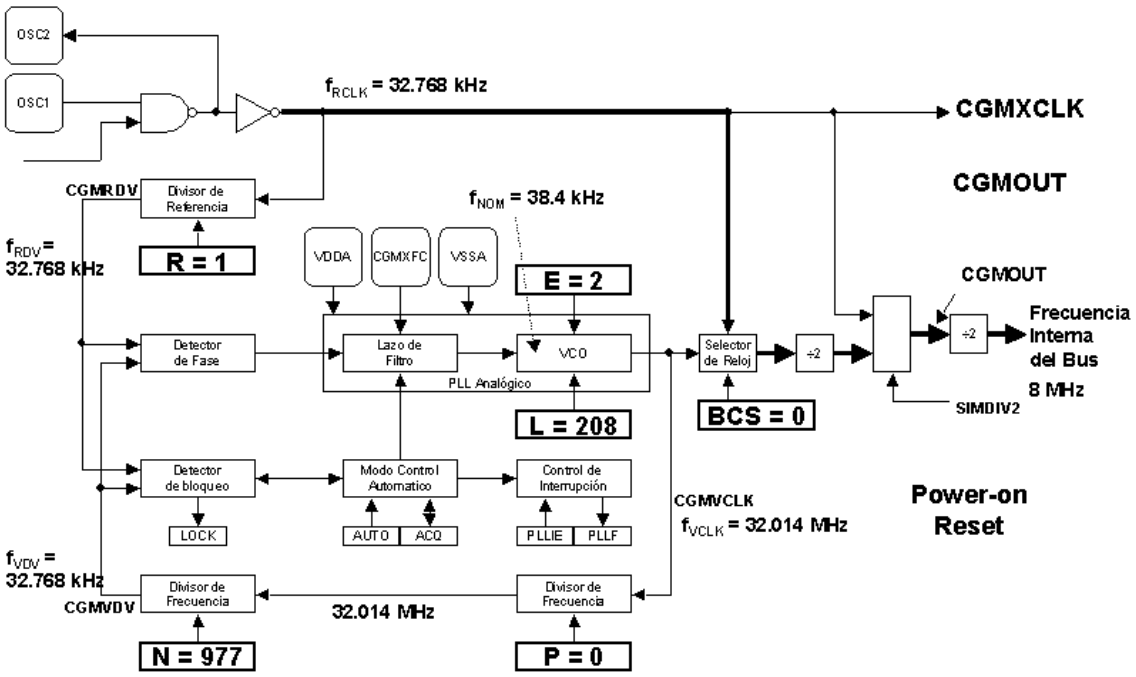
Frecuencias Precalculadas

Motorola proporciona algunos ejemplos en una tabla para frecuencias de bus diferentes, usando un cristal típico de 32.768 kHz. Esto hace fácil de programar el PLL para frecuencias de bus típicas. Los valores ya están en anotación hexadecimal.

f_{BUS}	f_{RCLK}	R	N	P	E	L
2.0 MHz	32.768 kHz	1	F5	0	0	01
2.4576 MHz	32.768 kHz	1	12C	0	1	80
2.5 MHz	32.768 kHz	1	132	0	1	83
4.0 MHz	32.768 kHz	1	1E9	0	1	D1
4.9152 MHz	32.768 kHz	1	258	0	2	80
5.0 MHz	32.768 kHz	1	263	0	2	82
7.3728 MHz	32.768 kHz	1	384	0	2	C0
8.0 MHz	32.768 kHz	1	3D1	0	2	D0

Generación de la Frecuencia de Bus

Ahora se puede ver cómo el CGM genera la frecuencia de bus de 8MHz.



Se han programado los registros con los valores para R, N, P, E y L, que se han calculado en el ejemplo anterior. La frecuencia del reloj, f_{RCLK}, es de 32.768 kHz. Esta va al divisor de referencia, que divide f_{RCLK} por el factor R=1 para que la frecuencia del reloj de referencia f_{RDV}, sea 32.768 kHz. La frecuencia nominal del centro de rango del VCO, f_{NOM}, es 38.4 kHz. La frecuencia del reloj de salida del VCO, f_{VCLK} está afectada por el factor lineal L=208 y por el factor de potencia dos E=2. Después de varios pasos, f_{VCLK} se pone a 32.014 MHz. Entonces f_{VCLK} pasa por un prescaler de potencia dos P=0, con f_{VCLK} dividido por 2P es 32.014MHz. Ahora el reloj del VCO se reduce por un factor N, consiguiendo f_{VDV} = 32.768 kHz. Esta frecuencia se compara con el reloj de referencia, f_{RCLK}. Una vez se igualan, el PLL se bloquea. Con el bit BCS puesto a 1, la frecuencia interna del bus objetivo será de 8 Mhz.

Después de un 'Power-On-Reset', el reloj del VCO no se bloquea y el bit BCS se pone a 0. Este selecciona el CMGXCLK como el reloj base, que es de 32.768 kHz. Entonces, hasta que la señal del reloj del VCO se bloquea, la frecuencia del bus es de 8.192 MHz en lugar de la frecuencia objetivo de 8MHz.

Registros del CGMC

Seguidamente se discuten el conjunto de registros del CGMC, que incluye seis registros para configurar y supervisar su funcionamiento.

Registro PCTL de control del PLL

El registro PCTL se usa para programar los valores de P y E.

PCTL	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	PLLIE	PLLF	PLLON	BCS	PRE1	PRE0	VPR1	VPR0
Escribir:								
Reset:	0	0	1	0	0	0	0	0

- El *bit PLLIE* activa la interrupción del PLL, es un bit de lectura/escritura que permite al PLL generar una petición de interrupción cuando el bit LOCK conmuta y pone a 1 el indicador PLLF. Cuando el bit AUTO en el registro de control del ancho de banda del PLL (PBWC) se pone a 0, PLLIE no se puede escribir y leer como lógica 0. Un Reset borra el bit PLLIE.
- El *bit indicador PLLF* de interrupción del PLL, es un bit de sólo lectura que se pone a 1 siempre que el bit LOCK conmuta. PLLF genera una petición de interrupción si el bit PLLIE también se pone a 1. PLLF siempre se lee como lógica 0, cuando el bit AUTO en el registro de control del ancho de banda del PLL (PBWC) está a 0. El bit PLLF se pone a 0 cuando hay un reset o cuando se lee el registro PCTL.

• El *bit PLLON*, es un bit de lectura/escritura que activa el PLL y activa el reloj del VCO, CGMVCLK. PLLON no se puede borrar si el reloj del VCO está manejando el reloj base, CGMOUT (BCS = 1). Un reset pone este bit para que el lazo se pueda estabilizar en el momento que la MCU se está alimentando.

• El *bit BCS* de selección del reloj base, es un bit de lectura/escritura que selecciona la salida del oscilador a cristal, CGMXCLK, o el reloj del VCO, CGMVCLK, como fuente de CGMOUT. La frecuencia CGMOUT es la mitad de la frecuencia del reloj seleccionada. BCS no se puede poner a 1, mientras el bit PLLON está a 0. Después de una conmutación del bit BCS, se necesita tomar hasta tres ciclos CGMXCLK y tres ciclos CGMVCLK para completar la transición de una fuente de reloj a otra. Durante la transición, CGMOUT se mantiene en espera del evento. Un Reset borra el bit BCS.

• Los *bits PRE1 y PRE0* de programación del prescaler, son bits de lectura/escritura que seleccionan el multiplicador potencia de dos del prescaler, P. PRE1 y PRE0 no se pueden escribir cuando el bit PLLON se pone a 1. Un reset borra estos bits. El valor de P es típicamente 0 al usar un cristal de 32.768-kHz como reloj de referencia.

• Los *bits VPR1 y VPR0* de selección de rango de potencia dos, son bits de lectura/escritura que controlan el rango multiplicador de potencia dos E hardware del VCO que, junto con L, controlan la frecuencia del centro de rango hardware, f VRS. Estos bits no se pueden escribir cuando el bit PLLON está a 1. Un reset borra estos bits.

Registro PBWC de control del ancho de banda del PLL

El registro PBWC de control del ancho de banda del PLL, se usa para configurar el modo de selección del PLL. También se usa para indicar cuando el PLL se bloquea.

PBWC	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	AUTO	LOCK	$\overline{\text{ACQ}}$	0	0	0	0	Reservado
Escribir:								
Reset:	0	0	0	0	0	0	0	0

• El *bit AUTO* del control automático del ancho de banda, es un bit de lectura/escritura que selecciona el control del ancho de banda automático o manual. Al inicializar el PLL para el funcionamiento manual (AUTO = 0), borra el bit $\overline{\text{ACQ}}$ antes de activar el PLL. Un reset borra el bit AUTO.

• Cuando el bit AUTO se pone a 1, el *bit LOCK* es un bit de sólo lectura, que se pone a 1 cuando el reloj del VCO se bloquea (corriendo a la frecuencia programada). Cuando el bit AUTO se pone a 0, LOCK lee como lógica 0 y no tiene ningún significado. La escritura a este bit se reserva para "prueba", por lo que este bit siempre se debe escribir como 0. Un reset borra el bit LOCK.

• Cuando el bit AUTO se pone a 1, el *bit* $\overline{\text{ACQ}}$ del modo de adquisición, es un bit de sólo lectura que indica si el PLL está en modo adquisición o en modo rastreo. Cuando el bit AUTO se borra, $\overline{\text{ACQ}}$ es un bit de lectura/escritura que controla si el PLL está en modo adquisición o modo rastreo. En modo control automático del ancho de banda (AUTO = 1), el último valor escrito del funcionamiento manual se guarda en una posición temporal y se recupera cuando continua el funcionamiento manual. Un reset borra este bit y activa el modo adquisición.

Registros PMSH y PMSL, de selección de la parte alta y baja del multiplicador del PLL

El registro de selección del multiplicador del PLL, es un registro de dos bytes que sirven para programar el valor del multiplicador de frecuencia del VCO, **N**.

PMSH	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	0	0	0	0	MUL11	MUL10	MUL9	MUL8
Escribir:								
Reset:	0	0	0	0	0	0	0	0

PMSL	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	MUL7	MUL6	MUL5	MUL4	MUL3	MUL2	MUL1	MUL0
Escribir:								
Reset:	0	1	0	0	0	0	0	0

• Los *bits MUL11 a MUL0* de selección del multiplicador, son bits de lectura/escritura que controlan los bytes altos y bajos del módulo divisor de realimentación que seleccionan el divisor de frecuencia del VCO. Los bits 7 - 4 del byte alto no son implementados y siempre se leen como lógica 0. Escribiendo un valor $0000 a estos registros se configura el módulo divisor de realimentación, lo mismo que escribir un valor $0001. Un reset inicializa los registros a $0040 para un valor por defecto multiplicando el valor de 64.

Los bits de selección del multiplicador tienen una protección interna tal que no se pueden escribir cuando el PLL está activado (PLLON = 1).

Registro PMRS selector de rango del VCO del PLL

El registro PMRS que selecciona el rango del VCO del PLL contiene la información requerida para programar el multiplicador de rango lineal **L** del VCO que junto con **E**, controlan la frecuencia del centro del rango hardware, f_{VRS}.

PMRS	Bit 7	6	5	4	3	2	1	Bit 0
Leer: Escribir:	VRS7	VRS6	VRS5	VRS4	VRS3	VRS2	VRS1	VRS0
Reset:	0	1	0	0	0	0	0	0

• Los *bits VRS7 a VRS0* de selección del rango del VCO no se pueden escribir, cuando el bit PLLON del registro PCTL se pone a 1. Un reset inicializa el registro a $40 para un rango multiplicador, por defecto, con valor de 64.

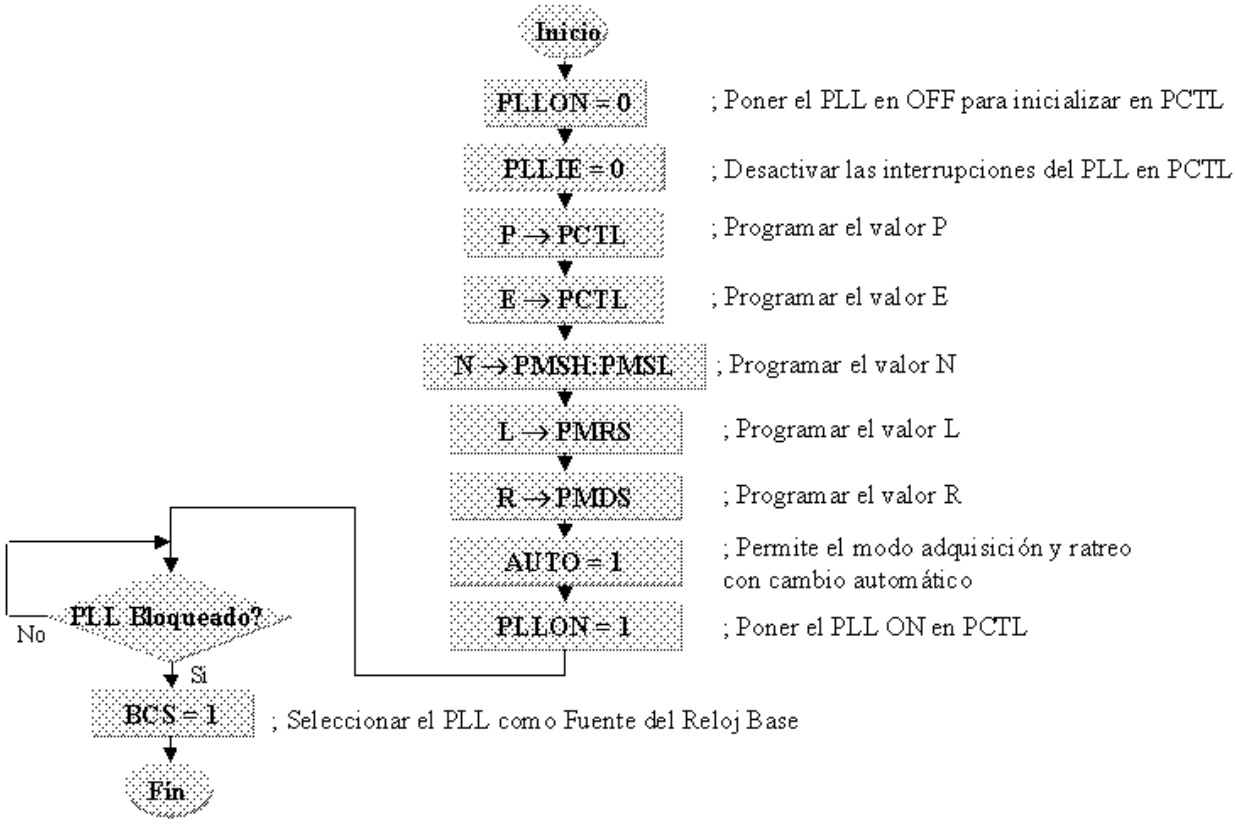
Registro PMDS selector del divisor de referencia del PLL

El registro PMDS de selección divisor de referencia, contiene la información de programación requerida para configurar el divisor de reloj de referencia, **R**.

PMDS	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	0	0	0	0	RDS3	RDS2	RDS1	RDS0
Escribir:								
Reset:	0	0	0	0	0	0	0	0

• Los *bits RDS7-RDS0* de selección del divisor de referencia, no se pueden escribir cuando el PLL está activado. Escribiendo un valor $00 en el registro PMDS, configura el divisor de referencia igual al valor $01. Un reset inicializa el registro a $01 para un valor del divisor predefinido de 1.

Diagrama de Flujo de Configuración del PLL



Primero se desactiva el PLL y las interrupciones, utilizando el modo consulta para verificar el bloqueo. Después se programan los valores P, E, N, L y R en los registros CGMC. Se pone el bit AUTO para permitir el filtro del lazo para cambiar automáticamente entre el modo adquisición y el modo rastreo. Se activa el PLL para alcanzar la frecuencia deseada. Se espera hasta que la frecuencia del VCO haya alcanzado el valor deseado, es decir, hasta la señal de bloqueo del PLL. Entonces se selecciona el PLL como la fuente de reloj base.

Ejercicio de Programación del PLL

Para completar esta parte se hace un ejercicio. Se escribe un programa para configurar el PLL del 68HC908GP32 PLL, para generar una frecuencia interna de bus de 8 MHz, o un oscilador de frecuencia de 32 MHz, usando un cristal de 32.768 kHz. También se escribe una subrutina para configurar el PLL y se usa el método de exploración del bloqueo. Para mostrar que la frecuencia interna del bus está a 8MHz, se puede hacer salir una señal cuadrada de 500 kHz a través del puerto C.

```
* -----------------------------------------------------------------
* Inicialización del 68HC908GP32
* -----------------------------------------------------------------
    MOV   #%00001011,CONFIG1          ; Configuración del registro CONFIG1
;                 \\\\\\\\_____________ Desactiva el módulo COP
;                  \\\\\\\_____________ Activa la instrucción STOP
;                   \\\\\\_____________ Modo Stop se recupera después de 4096 ciclos CGMXCLK
;                    \\\\\_____________ LVI trabajará a 5V
;                     \\\\_____________ Activa el módulo LVI
;                      \\\_____________ Activa el Reset de módulo LVI
;                       \\_____________ Desactiva LVI durante el modo STOP
;                        \_____________ COP periodo de tiempo 262,128 ciclos
    MOV   #%00000011,CONFIG2          ; Configuración del registro CONFIG2
;                 \\\\\\\\_____________ Se usa el Reloj Interno del Bus como fuente del SCI
;                  \\\\\\\_____________ Activa el Oscilador para operar durante Modo Stop
;                   \\\\\\_____________ Regulador de Tensión activado (Vdd > 3.6v)
;                    \\\\\_____________ Sin implementar
        CLRA                              ;Inicializa el Acumulador
        LDHX  #ENDRAM                     ;Stack Pointer -> Final de la RAM
        TXS                       ;       (H:X -> SP)
; FIN de la Inicialización de la MCU
```

Esta sección de código inicializa la MCU. Se puede usar esta sección como plantilla para todos los programas. El primer paso es inicializar la MCU usando los registros CONFIG1 y CONFIG2. Hay que recordar que estos registros sólo se pueden configurar una sola vez después de un reset, por lo que se recomienda ponerlo lo primero de todo al empezar un programa.

En este ejemplo, se desactiva el COP en el registro CONFIG1 para que no se tenga que alimentar el contador del COP para evitar un reset. También se activa la instrucción de STOP y se activa el LVI para trabajar a 5V. Usando el registro CONFIG2, se selecciona el reloj del bus interno como una fuente para la comunicación serie. También se activa el oscilador para trabajar durante el modo STOP, para que se pueda usar la característica de “wake-up” periódico. La configuración del registro CONFIG2 no es crítica para esta aplicación y se podría usar la inicialización predefinida.

La siguiente instrucción CLRA inicializa el acumulador para evitar advertencias en el "depurador". Las dos siguientes instrucciones redirigen el indicador de pila (SP) al final de la RAM. Hay que recordar esto para guardar compatibilidad con la familia HC05, la familia HC08 inicializa el indicador de pila automáticamente a $00FF, después de un reset.

```
* -----------------------------------------------------------------
* Aplicación
* -----------------------------------------------------------------

        JSR    ConfigPLL          ; Configura el PLL
        MOV    #$FF,DDRC          ; Activa el Puerto C como salida
Toggle:
        COM    PORTC              ; lazo de 8 ciclos
        NOP                       ; Conmuta el Puerto C cada 1µs (1MHz)
        BRA    Toggle             ; generando una señal cuadrada de 500kHz
* Fin de la Aplicación
```

Esta sección de código es donde realmente empieza el trabajo el PLL. La primera cosa que se debe hacer es configurar el PLL, saltando a una subrutina llamada ConfigPLL.

En la rutina principal, se pone el puerto C como salida. Para generar una onda cuadrada de 500 kHz, se necesita conmutar el puerto C a una velocidad de 1 MHz o una vez cada microsegundo. Corriendo a una frecuencia interna de 8MHz (0.125µs), se necesita un lazo de 8 ciclos para generar un retardo de 1µs. COM

PORTC es una instrucción de 4 ciclos, NOP es una instrucción de 1 ciclo y la instrucción BRA toma 3 ciclos, con un total de 8 ciclos. La rutina conmuta PORTC con la instrucción de complemento (COM) cada 8 ciclos.

```
* ------------------------------------------------------------------------------
* Subrutina para Configurar el que PLL explore a 32Mhz con un cristal de 32.768kHz
* ------------------------------------------------------------------------------
ConfigPLL:
        MOV    #%00000010,PCTL ; Configura el registro de control del PLL
;                    \  \  \\\\_________ Programa multiplicador potencia dos del VCO (E=2)
;                     \  \  \ \_________ Programa el divisor de Frecuencia Potencia de dos(P=0)
;                      \  \_____________ Desconecta el PLL, para que se pueda inicializar
;                       \________________ Desactiva las interrupciones del PLL
        MOV    #$03,PMSH       ; Programa el valor Modulo divisor Frecuencia del VCO, N
        MOV    #$D1,PMSL       ; en el registro de selección del Multiplicador VCO (N=0x03D1)
        MOV    #$D0,PMRS       ; Programa el valor multiplicador lineal del VCO, L
                               ; en el registro de selección de rango VCO del PLL (L=0xD0)
        MOV    #$01,PMDS       ; Programa el valor divisor de reloj de Referencia, R
                               ; en el registro selección de divisor del PLL (R=1)
        BSET   AUTO,PBWC       ; Permite modo adquisición y modo rastreo, automáticamente
        BSET   PLLON,PCTL      ; Conecta el PLL
        BRCLR LOCK,PBWC,*      ; Espera mientras PLL bloquea el VCO a la frecuencia deseada
        BSET   BCS,PCTL        ; Cuando bloquea, selecciona el PLL como fuente de reloj base
        RTS
* FIN de ConfigPLL
```

La subrutina ConfigPLL, se usa para configurar el PLL para generar una frecuencia de oscilación de 32MHz, o dividido por cuatro una frecuencia de bus de 8MHz. Se usan los valores que se calcularon en un ejemplo anterior, con R = 1, N = 0x03D1, P = 0, E = 2 y L = 0xD0.

Primero, se prepara el registro PCTL donde se programa E=2 y P=0. También, se desconecta el PLL para que se pueda inicializar y se desactivan las interrupciones del PLL, ya que se usará el modo consulta para verificar el bloqueo. Luego, se programa el valor N en el registro de selección del multiplicador del PLL y el valor L en el registro PMRS. También se programa el divisor de reloj de referencia, R, en el registro PMDS.

Luego, se pone el bit AUTO en el registro de control de ancho de banda del PLL (PCWC) para permitir cambiar automáticamente entre el modo adquisición y modo rastreo. Éste es el modo recomendado para la mayoría de las aplicaciones. Finalmente, se conecta el PLL y se espera mientras el PLL bloquea la frecuencia del VCO deseada, que en este caso es 32MHz. En cuanto el PLL haya bloqueado la frecuencia, se debe seleccionar el PLL como fuente de reloj base.

Módulo SCI (Serial Communications Interface)

El SCI es el módulo de interface de comunicación serie, ahora se describen las características y la configuración del Módulo SCI para la transmisión y la recepción de datos. También se preparan las comunicaciones serie del MC68HC08 a un PC.

Características del Módulo SCI

El Módulo SCI, también llamada UART (Universal Asynchronous Receiver Transmiter), proporciona comunicaciones con dispositivos periféricos u otras MCUs, en modo full-duplex, asíncrono y hasta gran velocidad. El SCI usa un formato estándar de no retorno a cero (NRZ) e incluye un generador de velocidad de transmisión (‘baud rate’) interno. Este generador del SCI interno no requiere del 'Timer' del sistema del 68HC08. En cambio, deriva una de las 32 frecuencias de ‘baud rate’ normales, directamente del oscilador de la MCU.

Se puede seleccionar una longitud del carácter de ocho o nueve bits. La configuración más común que se usa es un bit de inicio, 8-bits de datos del carácter y un bit de paro. La configuración de nueve bits usa un bit de inicio, 9-bits de datos del carácter y un bit de paro. El noveno bit de datos del carácter se puede usar como un bit de paro extra, como la función de ‘wake-up’ o de despertar el receptor del SCI o como un bit de paridad.

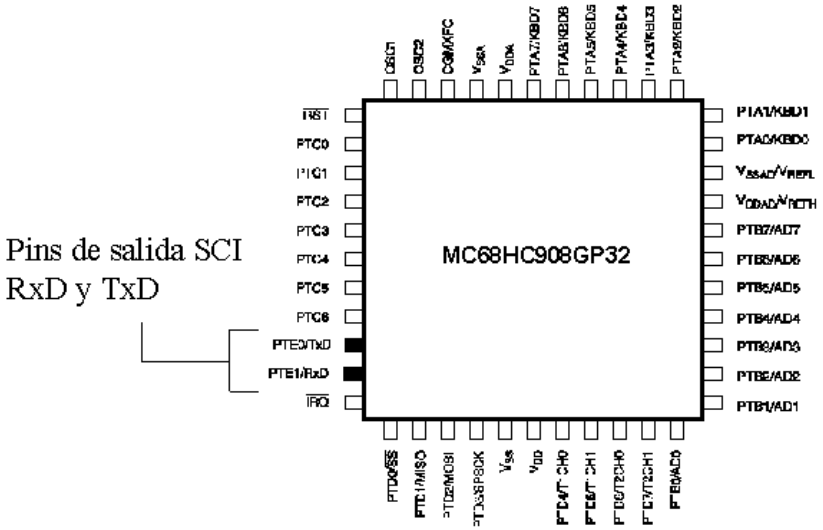
Se proporcionan dos métodos para despertar el receptor: ‘wakeup’ por dirección “marca” y el ‘wakeup’ por línea ocupada. En sistemas de múltiples receptores, este rasgo permite al Módulo SCI ignorar transmisiones predeterminadas a otros receptores entrando en un modo 'standby' o estado de espera.

Se activa el transmisor y receptor separadamente y son funcionalmente independientes, aunque usan los mismos formatos de datos y de velocidad de transmisión. El transmisor y el receptor son 'doble-buffered'. Esto significa que los caracteres “back to back” se manejan fácilmente, aun cuando la CPU tarda en responder al completar un carácter. Además, el receptor del SCI incluye varios rasgos avanzados para asegurar la recepción de datos muy fiable. Estos rasgos ayudan al desarrollo de redes de comunicación eficazmente.

Se puede programar la polaridad de salida del transmisor poniendo un bit en uno de los registros de control del SCI. Este rasgo le permite transmitir a nivel TTL. Algunos dispositivos transforman los niveles lógicos de TTL en niveles de datos estándar RS-232. Tales dispositivos requieren una polaridad de señal, en la que un nivel lógico 0 representa 0V y un nivel lógico 1 representa un voltaje por encima de 3V.

El SCI se puede interrumpir por medio de ocho indicadores ('flags') y peticiones de interrupción. El SCI mantiene las peticiones de interrupción y vectores, separados para el transmisor, el receptor y para las condiciones de error. Esto proporciona un proceso de interrupción muy eficaz, de función normal de transmisor/receptor sin ninguna exploración o verificación de interrupción. Cualquier condición de error se puede manejar por una rutina de servicio de interrupción separada.

La figura siguiente muestra la asignación de los pins del SCI en el MC68HC908GP32 MCU en encapsulado 44 QFP.



El Módulo SCI usa dos pins de entrada/salida. El pin RxD es un pin de entrada para recibir datos y el pin TxD es una salida para transmitir datos. Estos dos pins son compartidos con los pins del puerto de entrada/salida de la MCU. Cuando se desactiva el SCI, los dos pins se pueden usar como dos pins de entrada/salida de propósito general.

Si la aplicación requiere niveles RS-232 o RS-422, se necesita poner circuitos para cambiar a los niveles lógicos requeridos. Una transformación típica es de un nivel lógico 0 a 3 o 5V a niveles +/- 12V.

Registros del SCI

El conjunto de registros del Módulo SCI incluyen varios registros para controlar y verificar los funcionamientos del SCI, incluyendo un registro de datos, un registro de velocidad de transmisión, tres registros de control y dos registros de estado. Viendo cada registro, se puede ver cómo transmite y recibe datos el SCI usando el registro de datos del SCI.

Seguidamente se verá cómo se calcula la velocidad de transmisión usando la información del registro de velocidad de transmisión del SCI.

Registro SCDR de Datos del SCI

El registro SCDR de datos del SCI, realmente tiene dos registros que se acceden con una dirección. Los dos registros son el registro de datos recibidos y el registro de datos a transmitir.

SCDR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	R7	R6	R5	R4	R3	R2	R1	R0
Escribir:	T7	T6	T5	T4	T3	T2	T1	T0
Reset:	Un Reset no le afecta							

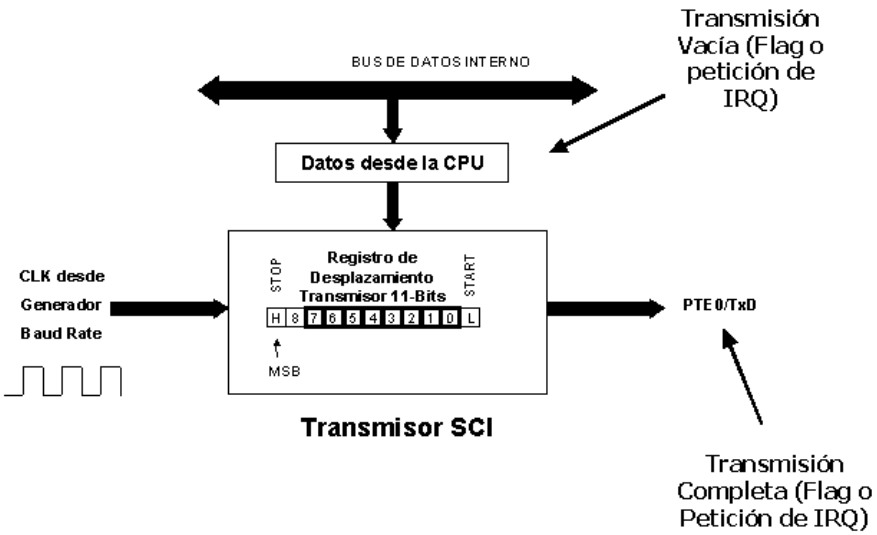
Se accede a los registros individualmente usando los comandos de lectura/escritura. Leyendo en el registro de datos del SCI, se leen los datos recibidos en el registro de datos. Escribiendo en el registro de datos del SCI, se escriben los datos a transmitir en el registro de datos.

Doble Buffer

El Módulo SCI incluye un 'buffer' doble, el del transmisor y el del receptor. A continuación se verá cómo esta característica se lleva a cabo en el transmisor del SCI.

Para transmitir datos, la CPU escribe datos al registro de datos del SCI. Además del registro de datos, el transmisor del SCI incluye un registro de desplazamiento de 11-bits para guardar los datos durante la transmisión. Esto le permite a la CPU escribir un carácter de datos mientras el SCI está transmitiendo el carácter anterior de datos. Para transmitir datos, la CPU sólo necesita verificar que el registro de datos del SCI está vacío antes de escribirlo.

Viendo un ejemplo típico de una transmisión de datos. Se asume que el registro de datos está vacío y la CPU quiere transmitir un carácter a través del transmisor del SCI. La CPU escribe el carácter en el registro de datos del SCI.

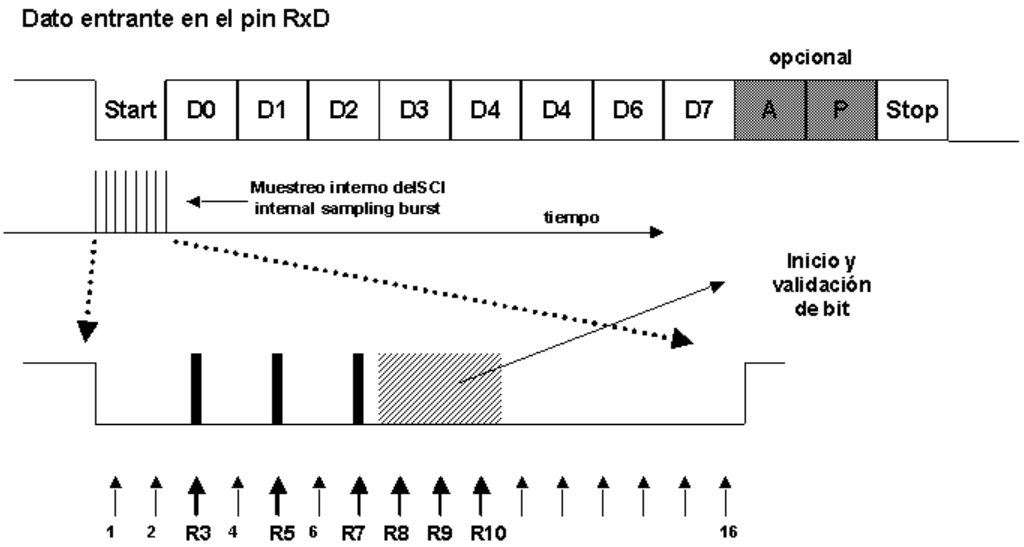


Cuando el registro de desplazamiento está listo para aceptar nuevos datos, el hardware del SCI mueve automáticamente los datos desde el registro de datos del SCI al registro de desplazamiento. Una vez los datos se mueven al registro de desplazamiento, el SCI pone un indicador de transmisor vacío para indicar a la CPU que otro carácter se pueda enviar. Se puede usar el software para consultar el indicador o se puede configurar el SCI para generar una petición de interrupción optativa.

El registro de desplazamiento se bloquea por el autogenerador de baudios y desplazan los 11 bits fuera del pin TxD. Una vez desplazado el último bit, el SCI pone el indicador de transmisión completa y opcionalmente genera una petición de interrupción para el proceso de la CPU. El receptor del SCI también usa un 'buffer' doble para recibir datos.

Recuperación de Datos del Receptor

El receptor del SCI incluye un sistema de recuperación de datos que prueba y verifica los datos. Esto asegura la recepción de datos muy fiable. Se puede ver en detalle cómo el receptor maneja un flujo de bits asíncrono que entra en el pin RxD.

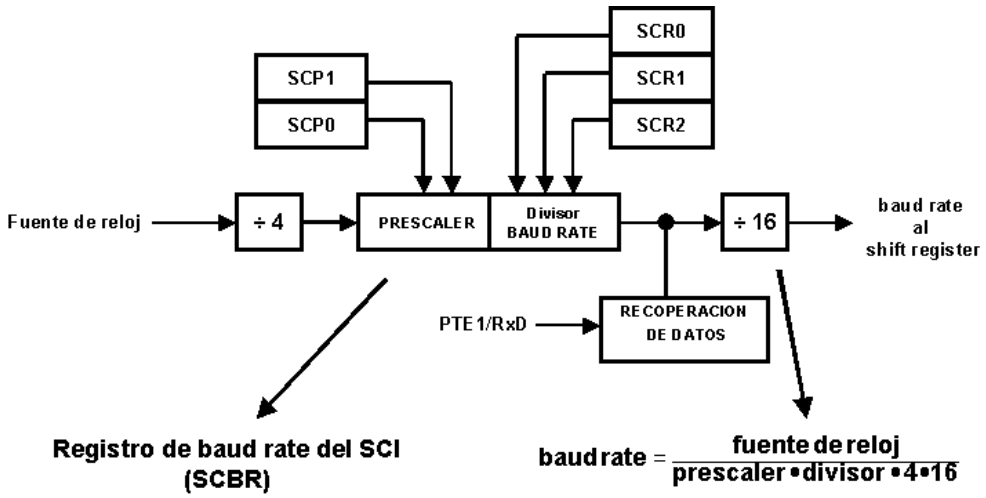


El sistema de recuperación de datos prueba el flujo de bits entrante a una velocidad fija de reloj, llamado RT, que es 16 veces la velocidad de transmisión. RT se sincroniza después de cada bit de inicio (‘start’) y después de cualquier bit de datos que cambia de nivel lógico 1 a 0. La sincronización adicional hace menos sensible, al módulo SCI, a las diferencias en las velocidades de transmisión del dispositivo enviado y la MCU. Para detectar un bit de ‘start’, el receptor hardware del SCI está ejecutando continuamente una búsqueda asíncrona para un nivel lógico 0 precedido por tres valores lógicos 1. Cuando ocurre un flanco de bajada de un posible bit de ‘start’, el reloj RT empieza a contar en R16.

La técnica de muestreo permite al SCI detectar posibles interferencias o ruidos que se solapan con el flujo de datos entrante. Para verificar la integridad del bit ‘start’ y para detectar el ruido, la lógica de recuperación de datos toma muestras en R3, R5 y R7 de RT. La verificación del bit ‘start’ falla si cualquiera de las dos de las tres muestras de la verificación, tiene un valor lógico 1. Cuando la verificación falla, el reloj RT se restablece y la búsqueda empieza de nuevo. Para determinar el valor de un bit de datos y para detectar el ruido, la lógica de recuperación toma muestras en R8, R9 y R10 de RT. Una situación similar ocurre cuando el receptor del SCI espera un bit de ‘stop’. En este caso, la lógica de recuperación toma muestras de R8, R9 y R10 de RT, para verificar la integridad del bit y para detectar el ruido.

Cálculo de la velocidad de transmisión del SCI

La figura siguiente muestra los bloques del generador de ‘baud rate’ del SCI. El generador de ‘baud rate’ incluye un prescaler y un divisor de ‘baud rate’ que se puede programar usando el registro de ‘baud rate’ del SCI.



La fuente reloj del generador de 'baud rate' varía según el dispositivo. La fuente de reloj del SCI del 68HC08 puede ser la frecuencia del cristal CGMXCLK dividido por cuatro o puede ser la frecuencia del bus interno en dispositivos con PLL o generador de reloj interno. Hay que verificar en el libro de datos técnicos del dispositivo, la aplicación específica del SCI.

El divisor por 4 del reloj de entrada es para bajar la velocidad a propósito. El divisor por 16 que va al registro de desplazamiento ('shift register') permite al receptor realizar el muestreo de datos a 16 veces la velocidad de 'baud rate'. Para calcular el 'baud rate' del SCI, hay que dividir la fuente de reloj por el producto del prescaler, el divisor de 'baud rate', el divisor por 4 y el divisor por 16.

Registro SCBR de Velocidad de Transmisión

Se puede programar el prescaler y el divisor 'baud rate' usando el registro de 'baud rate' del SCI, SCBR. Se deben seleccionar los valores apropiados para cada aplicación.

SCBR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:								
Escribir:			SCP1	SCP0	Reser-vado	SCR2	SCR1	SCR0
Reset:	0	0	0	0	0	0	0	0

• Los *bits SCP1 y SCP0* de 'baud rate' del prescaler del SCI, seleccionan uno de los cuatro valores del prescaler, como el que se muestra en la tabla.

SCP1 – SCP0	PRESCALER
00	1
01	3
10	4
11	13

• Los *bits SCR2 a SCR0* de selección de 'baud rate' del SCI, seleccionan uno de los ocho divisores, como el que se muestra en la tabla.

SCR2 a SCR0	DIVISOR DE BAUD RATE
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Cálculo del Baud Rate

Para generar un 'baud rate' de 19.200 baudios asumiendo una fuente de reloj con una frecuencia de 4.9152 MHz, se calcula de la siguiente manera:

$$\text{baud rate} = \frac{\textit{Fuente de Reloj}}{\textit{prescaler} \times \textit{divisor} \times 4 \times 16} = \frac{4.9152 MHz}{1 \times 4 \times 4 \times 16} = 19200 \text{ baudios}$$

La fuente de reloj dividido por el producto del prescaler por el divisor de 'baud rate' por el divisor por 4 y por 16. En este caso, se selecciona como fuente de reloj 4.9152, los bits SCP1-SPC0 con 00 para un valor del prescaler de 1 y los bits SCR2 a SCR0 con 010 para un divisor de 'baud rate' de 4, dando como resultado 19.200 baudios.

Registros SCC1, SCC2 y SCC3 de Control del SCI

El SCI tiene tres registros de control SCC1, SCC2 y SCC3. Se pueden configurar en modo funcionamiento normal, usando los bits del 7 al 0 del registro SCC1, los bits del 3 al 0 del registro SCC2 y los bits del 7 al 4 del registro SCC3. También se puede configurar el SCI para generar ocho interrupciones diferentes, usando los bits del 7 al 4 del registro SCC2 y los bits del 3 al 0 del registro SCC3. Primero se configuran los funcionamientos normales del SCI y después las ocho fuentes de interrupción del SCI.

SCC1: El registro SCC1 contiene ocho bits de control de lectura/escritura que se borran con un reset. Los ocho bits configuran los funcionamientos normales del SCI.

SCC1	Bit 7	6	5	4	3	2	1	Bit 0
Leer: Escribir:	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
Reset:	0	0	0	0	0	0	0	0

- El *bit LOOPS*, permite el funcionamiento en modo lazo. Este modo se usa principalmente para pruebas y diagnósticos. En modo lazo, el pin RxD está desconectado del SCI y la salida del transmisor se conecta a la entrada del receptor. El transmisor y el receptor tienen que estar activados para usar el modo lazo.
- El *bit ENSCI,* activa el SCI y el generador de 'baud rate'.
- El *bit TXINV* invierte la polaridad de los datos a transmitir e invierte todos los valores a transmitir, incluyendo el bit de "ocupado" ('idle'), el de "rotura" ('break') y los bits "inicio/fin" ('start/stop').
- El *bit M*, selecciona la longitud del carácter del SCI. Se puede seleccionar una configuración de ocho bits o nueve bits. En la configuración de nueve bits, el noveno bit puede servir como un bit de STOP extra, como una señal de 'wakeup' del receptor o como bit de paridad.
- El *bit WAKE*, selecciona el modo 'wake-up', 'marca de dirección' o 'línea ocupada'. El receptor del SCI puede entrar en modo espera ('stand-by') para ignorar mensajes determinados que van a un receptor diferente. Si se selecciona la 'marca de dirección', un nivel lógico 1 en la posición del bit más significativo de un carácter recibido, despierta el SCI con tiempo suficiente para ver el primer carácter del siguiente mensaje. Esto se usa típicamente para reducir trabajo en la CPU, en redes multi-drop del SCI. En modo de línea ocupada ('idle'), el receptor detecta cuando ningún otro SCI está transmitiendo supervisando el pin RxD.
- El *bit ILTY*, de tipo línea ocupada, selecciona el punto del flujo de bits donde el SCI empieza a contar bits lógicos 1 como bits de carácter ocupado ('idle'). Con el bit ILTY a 1, el SCI empieza a contar después del bit de STOP. Con el bit ILTY a 0, el SCI empieza a contar después del bit de START.
- El *bit PEN*, inserta un bit de paridad en la posición del bit más significativo.
- El *bit PTY*, selecciona el tipo de paridad, impar o par, que el SCI genera y verifica. Poniendo el bit a 1 selecciona paridad impar y poniendo el bit a 0 selecciona paridad par.

SCC2: El registro SCC2 contiene ocho bits de control de lectura/escritura que se borran con un reset. Los bits del 3 al 0, configuran los funcionamientos normales del SCI.

SCC2	Bit 7	6	5	4	3	2	1	Bit 0
Leer: Escribir:	SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK
Reset:	0	0	0	0	0	0	0	0

- El *bit TE* activa el transmisor. Empieza la transmisión enviando un preámbulo de 10 o 11 bits con nivel lógico 1, desde el registro de desplazamiento del transmisor al pin TxD. Poniendo el bit TE a 0, terminará cualquier transmisión en curso antes de devolver a la condición lógica 1 de ocupado.
- El *bit RE* activa el receptor. Poniendo el bit RE a 0, desactiva el receptor pero no afecta a los bits indicadores de interrupción del receptor.
- El *bit RWU* 'wakeup' del receptor, pone el receptor en modo 'stand by'. Cuando el receptor está en modo 'stand by', las interrupciones del receptor se desactivan. El bit WAKE en el registro SCC1 determina si una entrada de una 'marca de dirección' o de un 'idle' pone al receptor fuera del modo 'stand by' y el bit RWU se borra.
- El *bit SBK* envía un 'break', el SCI transmite un carácter 'break' seguido por un bit lógico 1. Esto garantiza el reconocimiento de un bit de 'start' válido. Si el bit SBK permanece a 1, el transmisor transmite continuamente caracteres 'break' sin bits lógicos 1 entre ellos.

SCC3: El registro SCC3 contiene ocho bits de control de lectura/escritura que se borran con un reset. Los bits del 7 al 4 configuran los funcionamientos normales del SCI.

SCC3	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE
Escribir:								
Reset:	U	U	0	0	0	0	0	0

U: NO afectado

• El *bit R8* guarda el octavo bit recibido, cuando el SCI está recibiendo caracteres de 9-bits (MSB 8 a LSB 0). El bit R8 es recibido al mismo tiempo que el registro SCDR recibe los otros 8 bits.

• El *bit T8* es muy similar al bit R8. El bit T8 se usa cuando el SCI está transmitiendo caracteres de 9 bits.

• Los *bits DMARE y DMATE* se usan en las MCUs con DMA, para activar transferencias DMA y servicios de recepción de DMA. El módulo DMA no se incluye en todas las MCU del 68HC08. Escribiendo un 1 en los bit DMARE o DMATE, puede afectar adversamente el funcionamiento de la MCU.

• Los restantes bits en los registros SCC2 y SCC3 configuran el SCI para generar ocho peticiones de interrupción diferentes de la CPU, cuando los bits de estado correspondientes se ponen a 1 en el registro de estado SCS1 del SCI. Estos bits de control se verán en los registros de estado.

SCS1: El registro de estado SCS1 del SCI, es un registro de sólo lectura que contiene ocho indicadores de estado para supervisar las operaciones del SCI.

SCS1	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	SCTE	TC	SCRF	IDLE	OR	NF	NE	PE
Escribir:								
Reset:	1	1	0	0	0	0	0	0

• El *bit SCTE* de transmisión vacía del SCI, se pone a 1 por el transmisor, cuando el registro SCDR transfiere un carácter al 'shift register' del transmisor. Si el bit SCTE del registro SCS1 se pone a 1 y el bit SCTIE del registro SCC2 está a 1, se generará una interrupción.

• El *bit TC* de transmisión completa, se pone a 1 cuando el bit SCTE se pone a 1 y no se está transmitiendo ningún dato, el preámbulo o el carácter 'break'. Si el bit TC se pone a 1, se generará una interrupción si el bit TCIE del registro SCC2 se pone a 1.

• El *bit SCRF* de receptor lleno del SCI, se pone a 1 cuando el dato del registro de desplazamiento del receptor se transfiere al registro SCDR. Si el bit SCRF se pone a 1, se generará una interrupción si el bit SCRIE del registro SCC2 se pone a 1.

• El *bit IDLE* de receptor ocupado, se pone a 1 cuando aparecen 10 u 11 valores lógicos 1 consecutivos en la entrada del receptor. Si el bit IDLE se pone a 1, se generará una interrupción si el bit ILIE del registro SCC2 se pone a 1.

• El *bit OR* 'overrun' del receptor, se pone a 1 cuando el software no lee el registro SCDR anterior, del registro de desplazamiento del receptor, recibiendo al siguiente carácter. Cuando pasa esto, se pierde el dato del registro de desplazamiento, pero el nuevo dato en el registro SCDR no se ve afectado. Si el bit OR se pone a 1, se generará una interrupción si el bit ORIE del registro SCC3 se pone a 1.

• El *bit NF* es un indicador de ruido del receptor, se pone a 1 cuando el SCI detecta ruido en el pin de entrada del receptor (pin RxD). Si el bit NF se pone a 1, se generará una interrupción si el bit NEIE en registro SCC3 se pone a 1.

• El *bit FE* de error de cuadro ('frame') del receptor, se pone a 1 cuando es aceptado un valor lógico 0 como el bit de 'stop'. Para borrar el bit FE, se lee SCS1 con el bit FE puesto a 1 y después se lee el registro SCDR. Si se pone el bit FE a 1, se generará una interrupción si el bit FEIE del registro SCC3 se pone a 1.

• El *bit PE* de error de paridad del receptor, se pone a 1 cuando el SCI detecta un error de paridad en los datos entrantes. Para borrar el bit PE, se lee SCS1 con el bit PE puesto a 1 y después se lee el registro SCDR. Si se pone a 1 el bit PE, se generará una interrupción si el bit PEIE del registro SCC3 se pone a 1.

SCS2: El registro de estado SCS2 del SCI, es un registro de sólo lectura que contiene dos indicadores de estado para supervisar los funcionamientos del SCI.

SCS2	Bit 7	6	5	4	3	2	1	Bit 0
Leer:							BKF	RPF
Escribir:								
Reset:	1	1	0	0	0	0	0	0

• El *indicador BKF* de 'break', se pone a 1 cuando el SCI detecta un carácter 'break' en el pin de entrada del receptor (PTE1/RxD). El bit BKF no genera ninguna petición de interrupción.

• El *indicador RPF* de recepción en progreso, se pone a 1 cuando el receptor detecta un nivel lógico 0 durante el periodo de tiempo RT1 de búsqueda del bit 'start'. El bit RPF no genera ninguna petición de interrupción.

Ejemplo de un Programa de Transmisión: **Transmitir "Hola" a 9600,n,8,1 a un terminal:**

Este programa configura el SCI para la transmisión de datos. Se activa el transmisor del SCI y se desactiva el receptor del SCI. La velocidad de transmisión es de 9600 baudios. El programa simplemente envía la palabra "Hola" a un terminal o cualquier otro dispositivo capaz de entender comunicaciones asíncronas.

```
$include 'iogp20.inc'
        ORG     $FFFE
        FDB     Aplicación              ;Vector de Reset
        ORG     $B000
Aplicación:
        MOV     #$0B,CONFIG1            ;No COP, 5V, No detección de bajo voltaje
        MOV     #$02,CONFIG2            ;SCI Usa reloj interno bus datos como fuente de reloj
        MOV     #%00000011,SCBR         ;Selecciona 9600 Baudios con un reloj de 4.9152MHz
        MOV     #%01000000,SCC1         ;Activa SCI
        MOV     #%00001000,SCC2         ;Activa Transmisión, Desactiva el receptor
        LDA     SCS1                    ;Condición para borrar el bit vacío SCT Tx
MAIN:   MOV     #'H',SCDR               ;Envía "H"
        BRCLR   7,SCS1,*                ;Espera para TX
        MOV     #'o',SCDR               ;Envía "o"
        BRCLR   7,SCS1,*                ;Espera para TX
        MOV     #'l',SCDR               ;Envía "l"
        BRCLR   7,SCS1,*                ;Espera para TX
        MOV     #'a',SCDR               ;Envía "a"
        BRCLR   7,SCS1,*                ;Espera para TX
        MOV     #$0013,SCDR             ;Envía <CR>
        BRCLR   7,SCS1,*                ;Espera para TX
        JMP     MAIN
```

El programa principal contiene una serie de instrucciones MOV para enviar los caracteres a través del puerto serie. Después de cada instrucción MOV, el programa debe esperar hasta que el registro de desplazamiento transmite (Tx) y mueve el siguiente byte a ser transmitido del registro SCDR. Hay que recordar que el SCDR es un byte del 'buffer' entre el SCI y la CPU. Cuando el registro SCDR transfiere un carácter al registro de desplazamiento del receptor, el indicador SCTE en SCS1 se pone a 1. La instrucción BRCLR que sigue cada instrucción MOV espera para que esto suceda.

Ejemplo de un Programa de Recepción: **Recibir un "4" para encender y apagar un LED:**

Este programa configura el SCI para la recepción de datos. Se activa el receptor del SCI y se desactiva el transmisor del SCI. El programa espera para recibir un carácter. Cuando esto pasa, el lazo de la aplicación principal es notificado a través de la rutina de servicio de interrupción RxEvent.

```
$include 'iogp20.inc'
        ORG     $FFFE
        FDB     Aplicación              ;Vector de Reset
        ORG     EPROM
Aplicación:
        MOV     #$0B,CONFIG1            ;No COP, 5V, No detección de bajo voltaje
        MOV     #$02,CONFIG2            ;SCI usa reloj bus interno como fuente de reloj
        MOV     #%00000011,SCBR         ;Selecciona 9600 Baudios con un reloj 4.9152 MHz
        MOV     #%01000000,SCC1         ;Activa el SCI
        MOV     #%00100100,SCC2         ;Activa el Receptor, Desactiva el transmisor
        LDA     SCS1                    ;Condición para borrar bit vacío SCT Tx
        CLR     PORTC                   ;Pone PORTC = 0x00
        BSET    4,DDRC                  ;Pone PORTC Bit 4 como Salida.
        CLI
Main:   JMP     MAIN                    ;Lazo Aplicación
RxEvent:
        LDA     SCS1                    ;Borra el 'flag' Rx
        LDA     SCDR                    ;Lee el carácter entrante
        CMP     #'4'                    ;Si Acc = ASCII(4) entonces complementa
                                        ;el contenido de salida PORTC
        BNE     EndIRQ                  ;todo lo demás fin de IRQ
        COM     PORTC                   ;Complementa el contenido de la salida del PORTC
EndIRQ  RTI                             ;Fin de esta IRQ
        ORG     $FFE4                   ;Pone el receptor SCI completo (llega un nuevo carácter)
        DW      RxEvent                 ;Define la Rutina de servicio de Interrupción
                                        ;en la Tabla del Vector de IRQ
```

La rutina de servicio de interrupción le dice a la CPU que lea el 'buffer' de entrada SCDR y lo compara con el código ASCII 4. Si son iguales, la CPU complementa el contenido del PORTC. Este conmuta un LED conectado al PORTC 4.

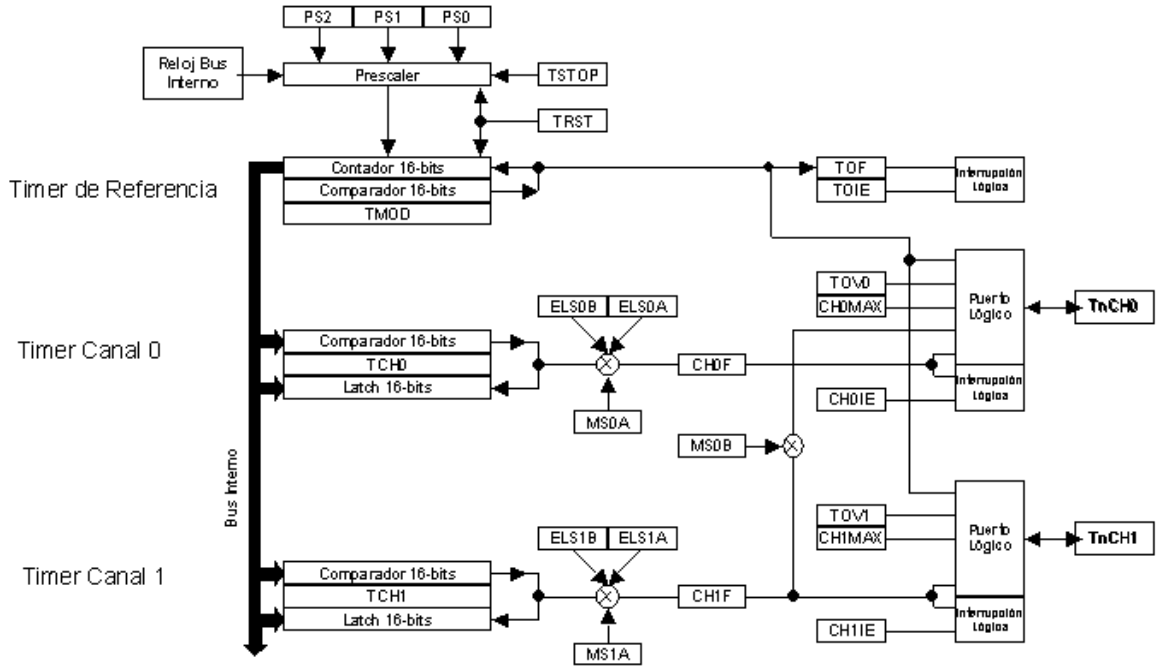
Módulo TIM (Timer Interface Module)

Esta parte cubre la configuración y el uso en detalle del TIM. Prepara y realiza una función de comparación de salida (‘output compare’), una función de captura de entrada (‘input capture’) y algunas funciones de modulación de ancho de pulso (PWM), mostrando las diferencias entre el PWM “unbuffered” (con perdida de tiempo en la recarga) y PWM “buffered” (sin perdida de tiempo por el valor cargado en un segundo registro).

Diagrama de Bloques del TIM

El TIM del 68HC08 tiene los canales del ‘Timer’ seleccionables por el usuario, en lugar de funciones específicas de ‘input capture’ (captura de entrada) y ‘output compare’ (comparación de salida). Cada canal es programable por el usuario para realizar una función de comparación de salida, una función de captura de entrada o una función de modulación de ancho de pulso. Estas funciones proporcionan flexibilidad para configurar el ‘Timer’ exactamente como se necesita para cada aplicación específica. También se pueden reconfigurar los canales en una aplicación para realizar otras tareas diferentes.

Cada 68HC08 tiene implementado con un módulo TIM con 2, 4, 6 o 8 canales de ‘Timer’ programables. Este diagrama de bloques solo tiene dos canales TIM.



Cada canal del ‘Timer’ comparte una sola referencia de tiempo, que viene de un módulo contador de 16-bits, un prescaler y un comparador. El prescaler es programable y proporciona, al contador común de referencia de 16 bits, un reloj que se deriva de la frecuencia del bus y que se divide por una de las siete posibilidades.

El módulo comparador permite al timer de 16-bits, contar sólo hasta el valor cargado en el registro del módulo comparador, TMOD, antes de un reset y reiniciando otra secuencia de conteo. Después se verá cómo esto puede ser útil en el generador PWM.

Se pueden obtener interrupciones por desbordamiento del contador y por cualquier actividad del canal del timer de entrada o de salida. Cada fuente de interrupción tiene su propio vector de interrupción, para permitir atender una interrupción sin perder tiempo analizando la fuente de la interrupción.

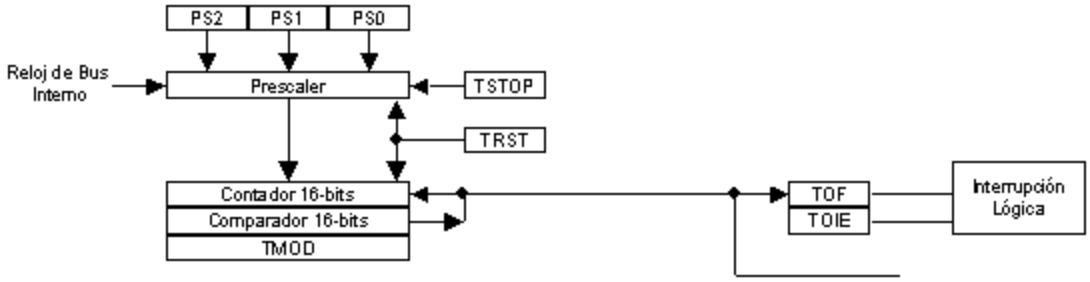
Tiempo de Referencia

La referencia de tiempo se genera por un contador de 16-bits, un prescaler y un módulo comparador. Para el trabajo normal, el contador de 16-bits corre continuamente como un contador libre. El contador se puede detener o se puede restablecer bajo el control del programa. Esto es particularmente útil cuando se quiere sincronizar la referencia del TIM con otro ‘timer’ o reloj.

El prescaler proporciona un valor de reloj derivado del bus interno del 68HC08, al contador de 16-bits y de un divisor programable. Usando PS0, PS1 y PS2, se puede programar el divisor por uno de los siete valores:

divido por 1, por 2 o por cualquier otra división binaria hasta por 64. También se puede seleccionar un reloj externo.

En su modo de más alta resolución (divido por 1), el TIM tiene una resolución de 125ns cuando está operando a la frecuencia de bus interno normal de 8MHz (1 dividido por 8 MHz). La referencia de reloj de 16 bits a una proporción más lenta, el prescaler puede dividir a menor frecuencia del bus interno, tanto como divido por 64.

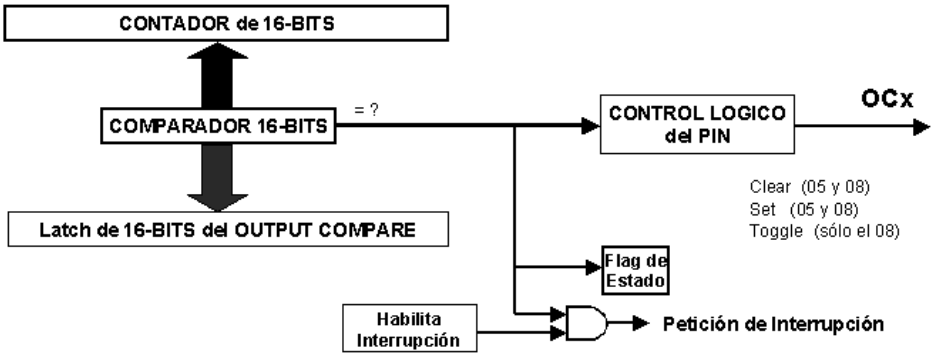


La referencia de tiempo de 16-bits incluye un registro TMOD, que se compara con el contador de 16-bits. Cuando se igualan, el indicador TOF de desbordamiento del 'timer' se pone a 1 y el contador de 16-bits se restablece y empieza otra secuencia de conteo. Por defecto, el registro del modulo se pone a *FFFFhex* que permite a la referencia contar a través de los 16-bits. Más adelante se verá como se puede escribir un valor diferente al registro del modulo para generar el periodo de una señal de modulación de ancho de pulso.

Se pueden cronometrar eventos más largos de 65,536 ciclos (16-bits), usando la función de desbordamiento del 'timer'. Cuando el indicador TOIE de interrupción de desbordamiento del 'timer' se pone a 1, se generará un evento de interrupción cada vez que el indicador TOF se pone a 1. Para cronometrar un evento de larga duración, se puede escribir código para contar el número de desbordamientos que ocurren. El prescaler mantiene la capacidad de optimizar la resolución versus el proceso de desbordamiento del 'timer'.

Función 'Output Compare'

La función de comparación de salida usa la referencia de tiempo de 16-bits, un comparador, un 'latch' de 16-bits de comparación de salida (output compare), un pin de salida, un pin de control lógico y una lógica de interrupción. La función de comparación de salida se puede ver con un ejemplo, un simple retardo.



Se empieza el ejemplo activando las interrupciones. Después, se selecciona un valor de 'offset' basado en el retardo deseado y se añade a este el valor del contador que corre libremente de 16-bits. Se escribe la suma del valor del contador y el 'offset' en el pin de comparación de salida. Cuando el contador que corre libremente se iguala al 'latch' de comparación de salida, se pone a 1 un indicador de estado, indicando que el retardo cronometrado ha ocurrido.

Opcionalmente, se puede poner un pin de salida a 1, a 0 o conmutando ('toggle') cuando el contador se iguala al valor de comparación del 'latch'. Se puede generar una interrupción opcional, permitiendo al usuario acercarse a tiempos muy precisos y crear eventos externos usando los pins de entrada/salida de comparación.

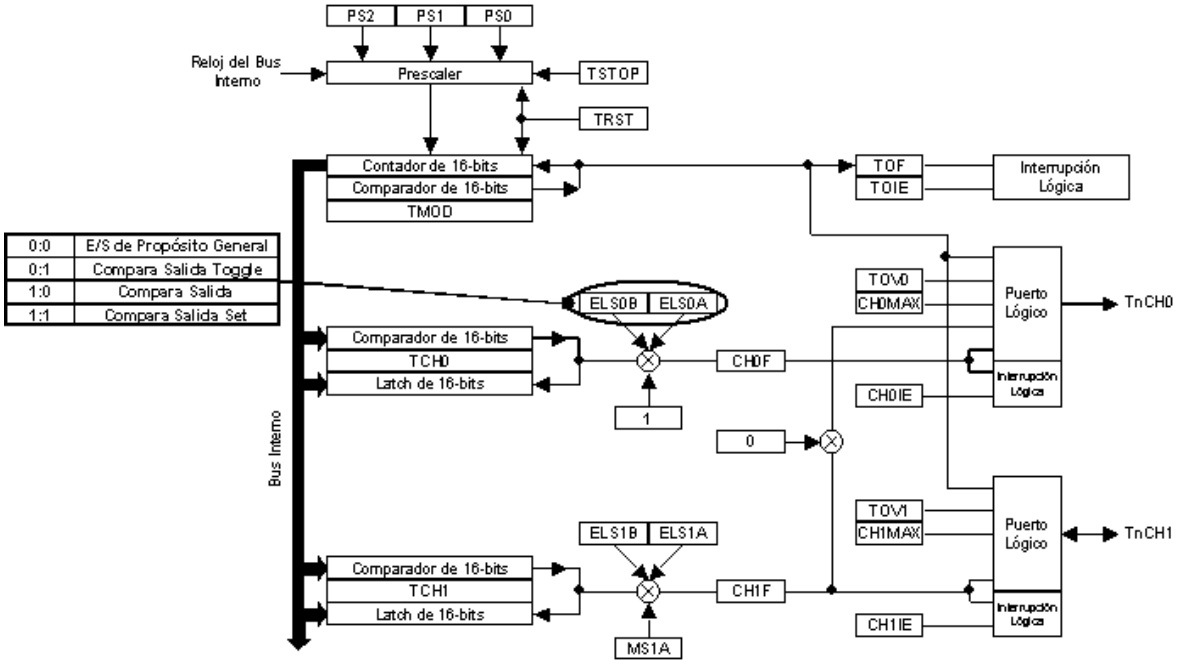
Aplicaciones 'Output Compare'

Se puede usar la función de comparación de salida, para una variedad de controles, incluyendo funciones comunes de cronometraje. En el ejemplo anterior, se usa la función de comparación de salida para realizar un simple retardo. Se puede extender el ejemplo, para realizar una interrupción periódica activando la interrupción de comparación de salida. Cuando la rutina de servicio recibe una interrupción de comparación de salida, calcula un nuevo valor de comparación de salida antes de continuar el proceso normal.

Usando el pin de salida puesto a 1, a 0 y opcionalmente conmutando ('toggle'), se puede implementar un simple pulso con ancho variable. Usando las mismas técnicas, se puede generar una frecuencia de salida variable o una señal de modulación de ancho de pulso (PWM). Más adelante se verán estas técnicas.

Aplicación de la Función 'Output Compare'

Se puede ver un ejemplo más detallado de una función de comparación de salida (output compare) usando el Canal 0. Se empieza poniendo MS0B a 0 para aislar los dos canales del timer y poniendo MS0A a 1 para seleccionar la función de salida. Entonces se utiliza ELS0A y ELS0B para programar la operación del pin.

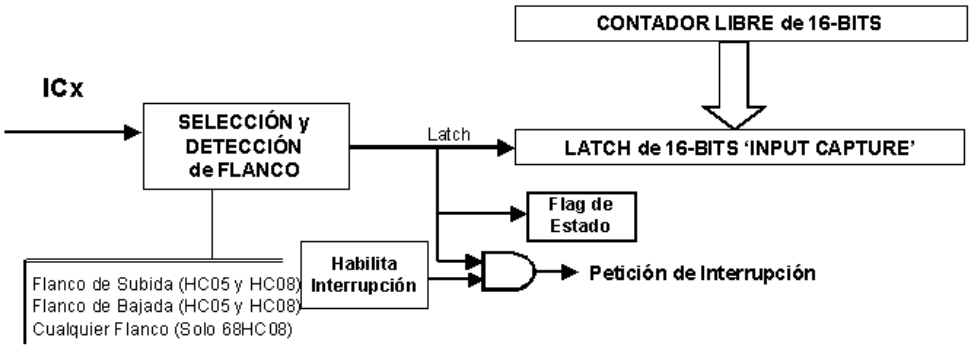


La opción 'toggle' se puede usar para simplificar la generación de pulsos. La comparación de salida (output compare) se usa para generar señales temporizadas. El valor guardado en TCH0 se está constantemente comparando con el contador de 16-bits. Cuando el valor en el contador de 16-bits se iguala a TCH0, la transición de la señal de salida programada (ELS0B:ELS0A) se fuerza en el pin TxCH0. Esto también puede generar una interrupción del canal del timer, si estaba activada.

Típicamente, se usará una rutina de servicio de interrupción, para fijar la próxima señal de salida del timer. Simplemente se lee el valor en TCH0, se añade un tiempo de 'offset' y se guarda el resultado en TCH0. Una vez más, en cuanto el valor del contador se ponga igual a TCH0, ocurre la actividad programada. Se da por supuesto que el contador corre libremente, que cuenta hasta *FFFFhex* y vuelve a cero. Si no, se tiene que tener en cuenta esto, al calcular el nuevo valor para escribir a TCH0.

Aplicación de la Función 'Input Capture'

La función de captura de entrada, se hace a través de un pin de entrada con selección de flanco, detector lógico y lógica de interrupción, usa el contador de 16 bits que corre libremente y el 'latch' del 'input capture' de 16-bits. Esta función permite cronometrar eventos externos para ser referenciados al contador que corre libremente de 16-bits.



Se puede configurar el pin de la entrada para buscar un flanco de subida, un flanco de bajada o cualquier tipo de flanco. En la figura se puede ver que sólo el 68HC08 usa el detector tanto de subida, como de bajada, como cualquier flanco, a diferencia del 68HC05. En este ejemplo se selecciona el flanco de subida. También se activan las interrupciones. Una vez se ha detectado el flanco seleccionado, el latch del 'input capture' se carga con el valor del contador que corre libremente y graba un tiempo de cuando ocurrió el evento. Se pone a 1 el indicador de estado y se puede generar una interrupción opcionalmente.

Aplicaciones 'Input Capture'

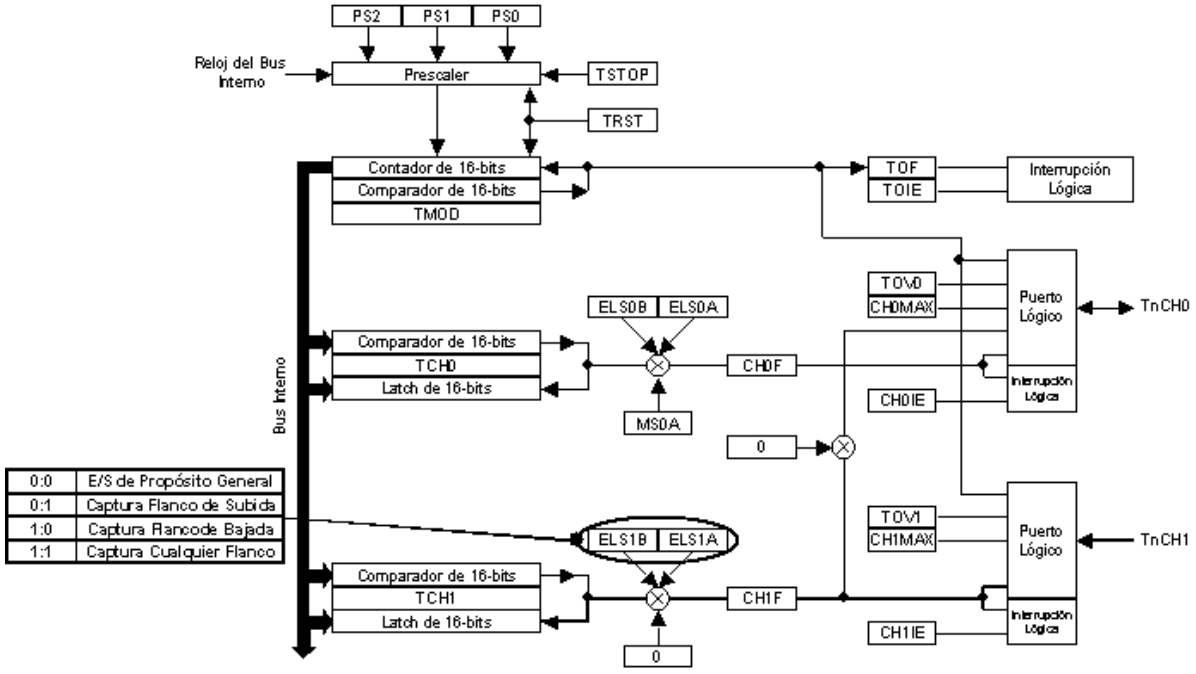
Se puede usar la función de captura de entrada ('input capture') para una variedad de funciones de control. En el ejemplo anterior, se usa la función de captura de entrada ('input capture') para realizar una referencia de tiempo absoluto de un evento externo. También se puede medir un periodo de entrada, activando la interrupción de captura de entrada. Cuando la rutina de servicio recibe una interrupción de captura de entrada, guarda temporalmente una copia del 'latch' de captura de entrada y después vuelve al proceso normal hasta que el flanco seleccionado es detectado de nuevo. La rutina de servicio substrae los dos valores de la captura de entrada para determinar el periodo. Para periodos largos, la rutina de servicio se ajusta para el número de interrupciones de desbordamiento del timer que ocurren entre el primer y segundo flanco.

Usando el 'input capture', se puede medir el ancho de un pulso de entrada. Esta función es similar a medir un periodo, excepto que la segunda captura se configura para detectar el flanco de bajada en lugar del flanco de subida inicial. Para anchos de pulso muy cortos, se pueden usar los dos canales del timer para mirar la misma señal con un canal para detectar el flanco de subida y el otro canal para detectar el flanco de bajada. Esto permite hacer medidas por debajo de 125 ns.

Si no se necesita la función de captura de entrada, se puede usar el pin de captura de entrada, como una línea de interrupción adicional. Después se verá un ejemplo más detallado de la función de captura de entrada.

Aplicación de la Función 'Input Capture'

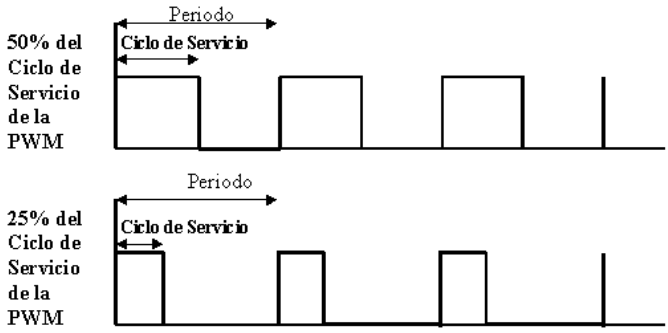
Se usa el Canal 1 para realizar una captura de entrada ('input capture'). Una vez más, se pone MS0B a cero, para aislar los Canales 0 y 1. También se pone MS1A a cero para seleccionar la función de entrada. ELS1A y ELS1B se usan para programar la operación del pin, se puede elegir cualquier forma de capturar un flanco, simplificando la medida del ancho de pulso, reduciendo el software.



Se usa el 'input capture' para medir señales cronometradas. El registro del canal del timer TCH1 se usa para poner el valor del 'latch' en el contador de 16-bits. Cuando ocurre la transición programada (ELS1B y ELS1A) en el pin (TxCH1), el valor del contador de 16-bits se guarda en el registro del canal del timer TCH1. Se puede generar opcionalmente una interrupción en la transición. Típicamente, se usa una rutina de servicio de interrupción para medir y calcular el tiempo entre transiciones de una señal, donde se determina el pulso o el ancho del periodo. Sabiendo el valor de dos transiciones, se puede calcular el pulso o el ancho del periodo.

Función PWM

La modulación del ancho de pulso, se usa para generar una forma de onda con un periodo fijo y el ciclo de servicio variable. El ciclo de servicio, es el tiempo relativo utilizado en los estados altos y bajos del periodo de la señal. Los moduladores de ancho de pulso pueden tener diferentes frecuencias y resoluciones. La frecuencia está definida por el periodo y la resolución es definida por el número de pasos discretos del ciclo de servicio que se pueden poner a 1.



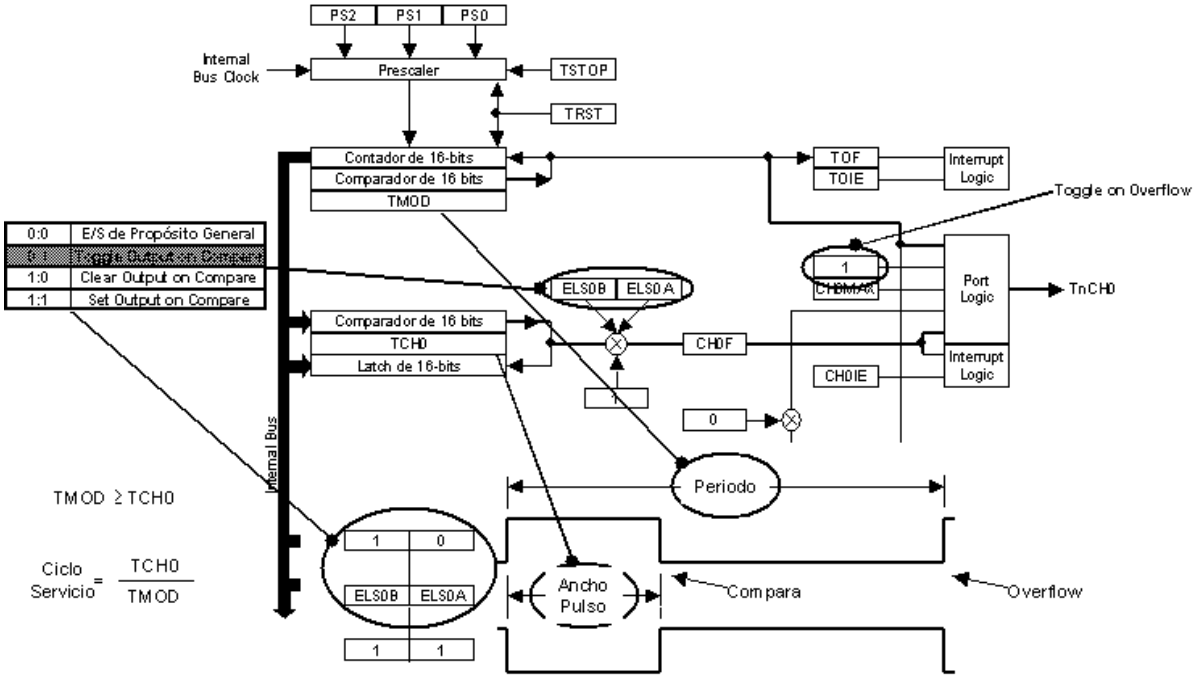
Un PWM de 8-bits permite especificar el ciclo de servicio en 256 pasos, es decir 2 elevado a 8. Por ejemplo, un ciclo de servicio de no-cero tan pequeño como 0.39% o lo que es lo mismo que 1 dividido por 256, podría especificarse con un PWM de 8-bits.

Un uso común del modulador del ancho de pulso, es la conversión digital a analógico usando algún filtro externo. El voltaje analógico generado es proporcional al ciclo de servicio. Teóricamente, un 50% del ciclo de servicio generará aproximadamente la mitad del voltaje analógico máximo y un 25% del ciclo de servicio generará un cuarto del voltaje analógico máximo.

El modulador de ancho de pulso también se usa para controlar normalmente un motor y para controlar la carga de una batería.

Función PWM 'Unbuffered'

Primero, se puede ver el procedimiento de puesta a punto e implementación de un PWM 'unbuffered' y después de un PWM 'buffered'. También se verán las diferencias entre los dos tipos.



El PWM 'unbuffered' opera como la función 'output compare'. Se muestra usando el Canal 0 del Timer. Así como con la función 'output compare', si se pone MS0B a 0 se aíslan los dos canales de los otros y si se pone MS0A a 1 se selecciona la función de salida. Por semejanza, se usa ELS0A y ELS0B para programar el funcionamiento del pin.

Hay que recordar que una señal PWM consiste en un Periodo y en un Ancho de Pulso. Para mayor flexibilidad, estos dos parámetros pueden ser programables.

Para generar la señal como la que se muestra en la figura, se configurará ELS0B:ELS0A como 1:0 o lo que es lo mismo "Borrar la Salida en Comparación". Después se usa el Registro del Canal del Timer (TCH0) para determinar el ancho del pulso.

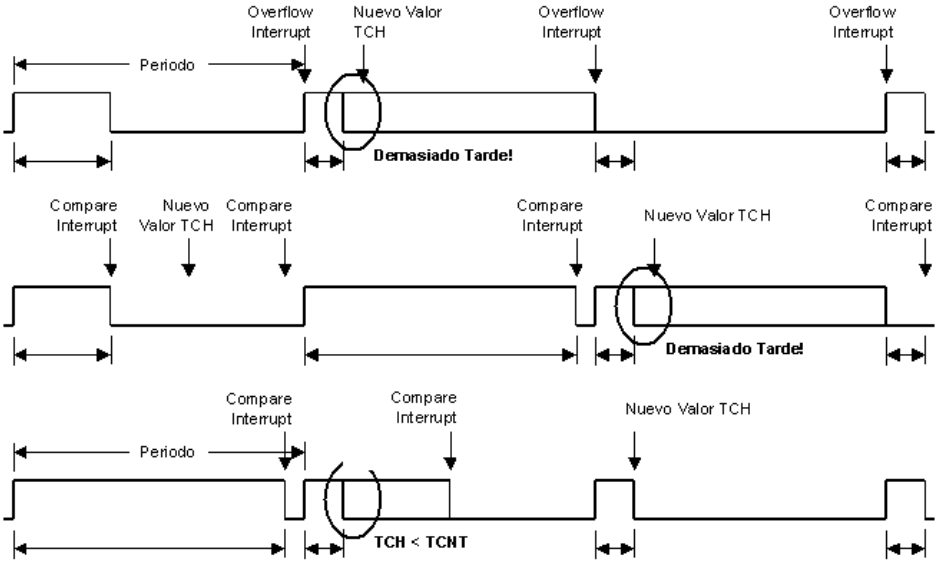
El Registro del Módulo Contador determina el periodo, obligando un desbordamiento del contador y restableciendo el Registro Contador cuando son iguales (por ejemplo, TCNT (contador de 16 bits) = TMOD).

Para entender la limitación del uso del PWM 'Unbuffered', se puede ver lo que pasa cuando se cambia el ciclo de servicio.

Como vio antes, para generar una forma de onda PWM como la que se muestra en la figura, se necesita preparar el canal del timer para borrar el ‘output compare’ y se activa la salida ‘toggle’ en función del desbordamiento del contador. Siempre se debe usar una rutina de servicio de interrupción para manipular el ancho del pulso para evitar señales espurias, causadas por la desincronización de escrituras en el registro de canal del timer. Las posibles elecciones son: interrupción por Comparación de Salida y por Desbordamiento del Contador.

Para los propósitos de este primer ejemplo, se ha elegido la interrupción de desbordamiento del Contador. Después, se puede ver lo que pasa cuando se usa la interrupción por desbordamiento del contador para cambiar el funcionamiento del PWM a un ciclo de servicio más pequeño. Véase que las formas de onda del ciclo de servicio mostrados en el diagrama no están dibujadas a escala.

Mientras que ocurre la petición del servicio de interrupción, en el momento de la transición de la señal de salida, tomará un tiempo para la rutina de servicio de interrupción para preparar el canal del timer para la siguiente operación de 'output compare'. Este retardo también es dependiente en si o no están en ese momento otras interrupciones en servicio. Para que en una aplicación que esté muy ocupada, este retardo sería casi imprevisible. Naturalmente, uno siempre debe considerar el peor caso posible.



En la figura mostrada, la rutina del servicio de interrupción por desbordamiento, toma demasiado tiempo para calcular y prepararse para la siguiente transición del canal del timer (por ejemplo, un 'output compare'). Ya que el valor en el contador ha superado el nuevo valor en el registro del canal del timer, no se ejecutará ninguna operación de 'output compare' y el estado del pin de salida no cambia.

Esto es así hasta la siguiente interrupción por desbordamiento del contador, hasta que tiempo el nivel de salida es ‘toggled’; en este caso, de alto a bajo. El funcionamiento del 'output compare' todavía se configura con el nuevo valor de ciclo de servicio. Sin embargo, ya que el funcionamiento programado es borrar el pin en la comparación, el estado del pin no cambia. De hecho, sólo después de la siguiente operación de desbordamiento del contador, se consigue finalmente el ciclo de servicio para el que se ha esta haciendo.

En resumen, al usar la rutina de servicio de interrupción por desbordamiento del contador para sincronizar modificaciones al ciclo de servicio del PWM, cambiando (en una aplicación específica) a un ciclo de servicio muy pequeño, causará (potencialmente) un funcionamiento indeseable. La señal experimentará 100% y después 0% ciclos de servicio antes de establecerse en el valor programado. Esto será así, cada vez que se haga un cambio a un ciclo de servicio muy pequeño. Por lo que, usando la interrupción por desbordamiento del contador quizá no es la mejor opción.

En cambio, se puede probar de usar la interrupción 'output compare' para sincronizar los cambios del ciclo de servicio del PWM. Como se puede ver, esta interrupción ocurrirá antes del desbordamiento del contador. Para los propósitos de este ejemplo, se cambiar el funcionamiento del PWM a un ciclo de servicio muy grande.

Así como antes, la petición del servicio de interrupción ocurre en el momento en que transiciones de las señales de salida y así tomará tiempo para la rutina del servicio de interrupción para preparar el canal del timer para la siguiente operación de 'output compare'. Pero este retraso debe ser antes del siguiente periodo de PWM.

De hecho, muy probablemente es que este cálculo y la preparación causará otra interrupción de 'output compare' para que ocurra dentro del mismo periodo de PWM. Esto no tiene ningún efecto adverso en la actividad del pin de salida y esta segunda petición de servicio de interrupción, seguramente se puede ignorar. Como se esperaba, el desbordamiento del contador hará conmutar (‘toggle’) la salida. Y el 'output compare' realizará la transición correcta.

¿Pero, que pasa cuándo se reprograma el PWM durante un ciclo de servicio muy pequeño?: Se descubrirá que se tiene el mismo problema que antes. Debido a la potencia de la rutina de servicio de interrupción la respuesta se retrasa, el cálculo para el nuevo (muy pequeño) ciclo de servicio podría tomar una inaceptable cantidad de tiempo muy grande.

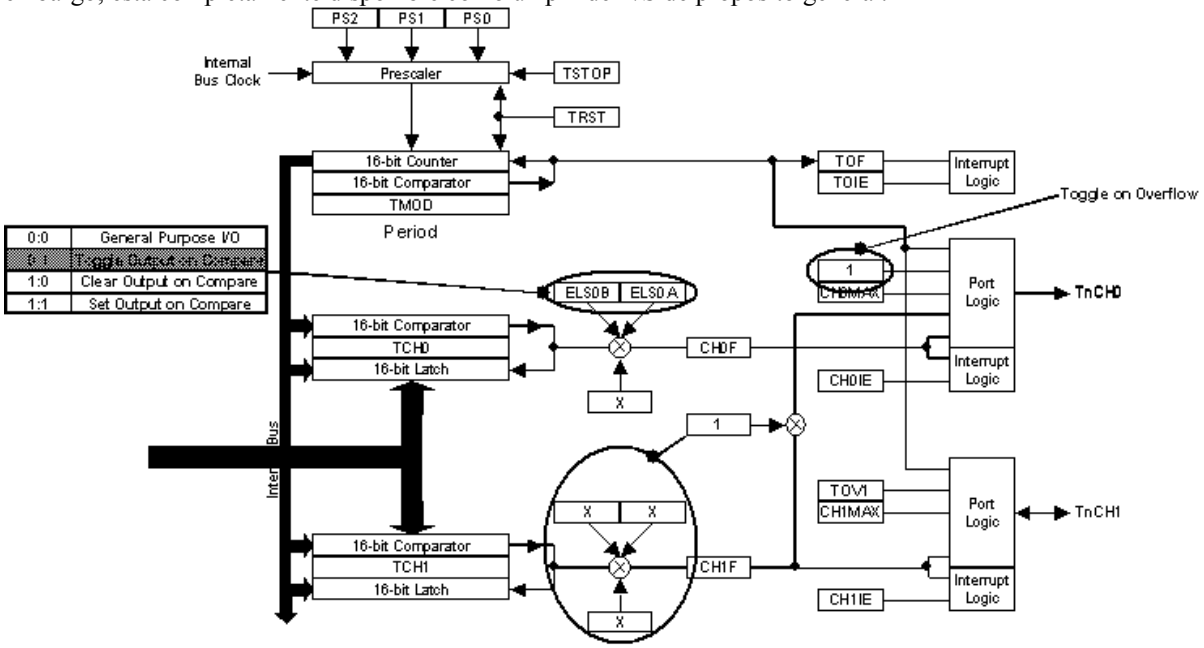
Y así, como se vio antes, la transición del funcionamiento PWM a través de un 100% y después del 0% del ciclo de servicio antes de establecerse en el valor programado. Esto será cierto, si se pide un cambio de un ciclo de servicio muy grande a un ciclo de servicio muy pequeño. Si la transición entre un pulso largo y un pulso corto, resulta en ninguna transición indeseable, asumiendo que es un pulso del 100% de ciclo de servicio seguido por un pulso del 0% de ciclo de servicio insertado entre ellos, analizando esta situación puede ser aceptable. Sin embargo, la clave está calculando la rutina de servicio de interrupción del 'output compare' con una respuesta de más largo tiempo que determina las condiciones de cuando esto ocurre. Dependiendo de la aplicación, un tiempo de respuesta más largo podría hacer este tipo de funcionamiento indeseable. Comparando las dos fuentes de interrupción, sólo usando la interrupción 'output compare' presenta un problema potencial cuando la transición del ciclo de servicio pasa de muy grande a muy pequeño. Esto es superior al usar la interrupción por desbordamiento del contador, tal como se vio en el funcionamiento anterior, cada vez que se hace una transición a un ciclo de servicio pequeño, sin tener en cuenta el valor anterior.

Investigando una posible solución a esta situación. Una vez más, se puede empezar con un ciclo de servicio muy grande. Y se escogerá la transición a un ciclo de servicio muy pequeño.

Usando la interrupción 'output compare', tal como se vio, el proceso de calcular el nuevo valor del ciclo de servicio, puede tomar demasiado tiempo. Esta situación puede ser detectada comparando el nuevo valor con otro valor actual del contador (probablemente sólo los 8-bits más bajos) o algún otro valor predeterminado, en el peor de los casos. En lugar de usar el nuevo ciclo de servicio pequeño, se escribirá un valor intermedio durante un ciclo que se fuerza a un 50% del ciclo de servicio. Ahora el siguiente ciclo (es decir, la rutina de servicio de interrupción 'output compare') programa el canal del timer con el ciclo de servicio pequeño intencionado. El tiempo requerido para realizar esta tarea es de antemano en el siguiente periodo de PWM. Y el funcionamiento aquí es como se deseó. Ahora, las transiciones del 99% al 50% al 1%, son probablemente aceptables en la mayoría de las aplicaciones.

PWM 'Buffered'

Para las situaciones donde el PWM unbuffered no trabajará, se puede escoger el modo Buffered que puede quitar cualquier posibilidad de espurios en señal de salida. Se requiere el uso de dos registros del canal del timer para controlar el manejo de la señal en el pin TxCH0. El pin TxCH1 no se puede usar por el timer; sin embargo, está completamente disponible como un pin de E/S de propósito general.



Los dos canales del timer se pueden unir funcionalmente, poniendo MS0B a 1. Poniendo MS0B a 1, obligará al pin TxCH0 que sea una salida, por lo que la configuración de MS0A no le importa. Además, poniendo MS0B a 1, activa ELS0A y ELS0B para ser usado para programar el funcionamiento del pin, tal como se ha visto antes.

Ya que la actividad del pin TxCH0 está controlada a través de ELS0A y ELS0B, los otros bits de control de canal del timer, ELS1A, ELS1B y MS1A no hay que tener cuidado.

Como en el modo PWM Unbuffered, el valor guardado en el registro del canal del timer, TCH0, determina el ciclo de servicio. Y el valor en TMOD dicta el periodo de señal PWM, con el contador constantemente comparado al valor en TCH0.

Cuando el valor en el contador de 16-bits es igual a TCH0, la transición de la señal de salida programada (por ejemplo, “Borrar la salida en Comparación”) se fuerza en señal en el pin TxCH0.

Una vez más, como en el caso de PWM Unbuffered, se debe programar el bit TOV0 (o “Toggle por desbordamiento”). Poniendo TOV0 a 1 conmutará (‘toggle’) la salida cada vez que haya desbordamientos del contador de 16-bits.

Cuando el valor en el contador de 16-bits es igual a TMOD, se fuerza un desbordamiento, restablecimiento el contador a cero y con TOV=1, el pin de salida conmuta (‘toggled’).

Después, para cambiar el ciclo de servicio, el software debe programar el siguiente valor en los registros del canal del timer alternativamente. Si TCH0 está actualmente activo, entonces cualquier cambio en el ciclo de servicio PWM se escribe en TCH1 y viceversa.

Se sincronizan los cambios del ciclo de servicio con el siguiente desbordamiento del contador, determinado por TMOD. Subsecuentemente, en este ejemplo, TCH0 está actualmente activo y se ha escrito un nuevo valor en TCH1, la lógica empezará automáticamente usando TCH1 durante el siguiente ciclo de servicio.

El siguiente cambio al ciclo de servicio requerirá una escritura en TCH0. El software del usuario necesitará guardar la traza de lo que fue escrito anteriormente en el registro del canal del timer, para determinar que canal será escrito a continuación. Pero, como que ahora se sincronizan automáticamente los cambios del ciclo de servicio con el timer, este software ya no necesita residir dentro de una rutina de servicio de interrupción del timer.

Cuando el valor en el contador de 16-bits es igual a TCH1, la transición de la señal de salida programada (por ejemplo, “Borrar la salida en Comparación”) se fuerza en el pin la señal TxCH0. Se generará el PWM deseado hasta un nuevo ciclo de servicio, que es programado en los registros TCH1 o TCH0.

Módulo TBM (Time Base Module)

En esta sección se describen las características y la configuración del TBM. También se puede ver cómo se configura el TBM para generar eventos periódicos, así como el auto-despertarse del modo STOP de bajo consumo. Sólo algunos 68HC908 contienen este módulo.

Posibilidades del TBM

• Puede generar interrupciones periódicas seleccionables por el usuario
  - Independientes del timer del sistema.
  - Directamente gobernado desde el cristal externo.
  - Programable por software a 1 Hz, 4 Hz, 16 Hz, 256 Hz, 512 Hz, 1024 Hz, 2048 Hz o 4096 Hz.
  - Puede trabajar como reloj en tiempo real de bajo consumo.
• Puede ser activado en modo STOP
  - Permite autodespertarse periódicamente del modo STOP.
  - Elimina circuitería externa.
  - Trabaja con muy bajo consumo, en microamperios.

Se puede usar el TBM del 68HC08 para generar interrupciones periódicas seleccionables por el usuario. Esto es útil para ejecutar diagnósticos periódicos, abastecer a los periféricos, realizar mantenimientos o cualquier otro evento fijado regularmente. Se puede usar la función de comparación de salida del 'timer' para generar una interrupción periódica, pero este método necesita incluir algún software, en cambio el TBM trabaja independientemente del sistema del 'timer' y no requiere ningún software. Usando el TBM se ahorran también canales disponibles del 'timer' para la aplicación.

La familia 68HC908GP y GR, el reloj del cristal externo va directamente al TBM y no de la salida del oscilador. Las MCU con el Generador del Reloj Interno (ICG), como la familia 68HC908KX, trabajan ligeramente diferente de lo que se describe en esta guía didáctica. Hay que verificar la documentación de cada producto sobre las diferencias específicas.

Cuando la fuente de reloj es un cristal de 32KHz, el TBM se puede configurar para generar interrupciones periódicas en una gama amplia de frecuencias entre 1 Hz y 4096 Hz. También, el TBM puede generar eventos de tiempo, como los que se usó en su momento en los relojes digitales. Con un simple software, se puede usar el TBM como un reloj de tiempo real de bajo consumo y bajo costo. Como ejemplo, el TBM se puede configurar para generar una interrupción una vez por segundo. La rutina de servicio de interrupción del TBM puede generar segundos, minutos, horas, días y años guardando las variables en la RAM o en la memoria no-volátil.

El TBM puede ser muy útil en aplicaciones de bajo consumo. Para minimizar consumo de corriente y aumentar la vida de la batería, el 68HC08 se puede poner en modo STOP para detener el oscilador interno. Por ejemplo, una aplicación alimentada con batería puede entrar en modo STOP para esperar una entrada del usuario, como la de un pequeño teclado. El 68HC08 incorpora una circuitería para una interrupción por teclado, eliminando resistencias y el resto de circuitería necesaria para llevar a cabo una interrupción por teclado. Otra aplicación alimentada con batería puede necesitar que se despierte la MCU periódicamente para verificar un estado, tomar una lectura de un sensor o realizar un diagnóstico antes de volver al modo STOP de bajo consumo. Durante el modo STOP, no trabaja ninguno de los periféricos internos, ya que no tienen ninguna fuente de reloj. Por consiguiente, normalmente se requiere una interrupción externa o un 'reset' para despertar la MCU de este modo y continuar el funcionamiento normal. El TBM permite operar directamente del cristal durante el modo STOP. Usando la característica de auto-despertarse del TBM, la CPU puede despertarse del modo STOP sin ninguna circuitería externa, reduciendo el costo del sistema.

Registro TBCR de Control del TBM

Ahora, se puede ver cómo se configura el TBM usando el registro de control TBCR del TBM.

TBCR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	TBIF	TBR2	TBR1	TBR0	0	TBIE	TBON	TBTST*
Escribir:					TACK			
Reset:	0	0	0	0	0	0	0	0

• El *bit TBON*, es un bit de lectura/escritura que activa el TBM. Cuando se necesita el TBM, se puede apagar el Módulo para reducir el consumo. El contador de TBM se puede inicializar poniendo a 0 y a 1 este bit. Un reset pone a 0 el bit TBON.

• El *indicador TBIF* de interrupción del TBM, es un bit de sólo lectura que se pone a 1 cuando en el contador hay un desbordamiento. Cuando se activan las interrupciones del TBM, un 1 en este bit indica que está pendiente una interrupción.

• El *bit TACK* de reconocimiento, es un bit de sólo escritura que siempre se lee como 0. Escribiendo un 1 en este bit, se borra el bit TBIF. Escribiendo 0 en este bit no tiene ningún efecto.

• El *bit TBIE*, es un bit de lectura/escritura que activa las interrupciones del TBM. Cuando este bit se pone a 1, el TBM genera una interrupción, cuando el bit TBIF se pone a 1. Un reset borra el bit BIE.

• Los bits *TBR2 a TBR0*, son bits de lectura/escritura que seleccionan la velocidad de las interrupciones periódicas. No se deben cambiar los valores de estos bits mientras se activa el TBM.

• El *bit TBTST* no está implementado.

Se puede usar el bit OSCSTOPEN del registro CONFIG para configurar el TBM para trabajar durante el modo STOP. Si el bit OSCSTOPEN se pone a 1, el TBM continúa siendo gobernado directamente por el cristal después de la ejecución de la instrucción STOP. Si este bit se borra, el TBM no trabajará en modo STOP.

Selección de la Base de Tiempos

La siguiente tabla muestra los bits (TBR2 a TBR0) para seleccionar la base de tiempos del TBM. Esta tabla está basada con un reloj externo de 32.768 KHz. Se puede ver, que se pueden generar frecuencias entre 1 Hz y 4096 Hz. Cambiando el reloj externo se pueden generar submúltiplos de estas frecuencias.

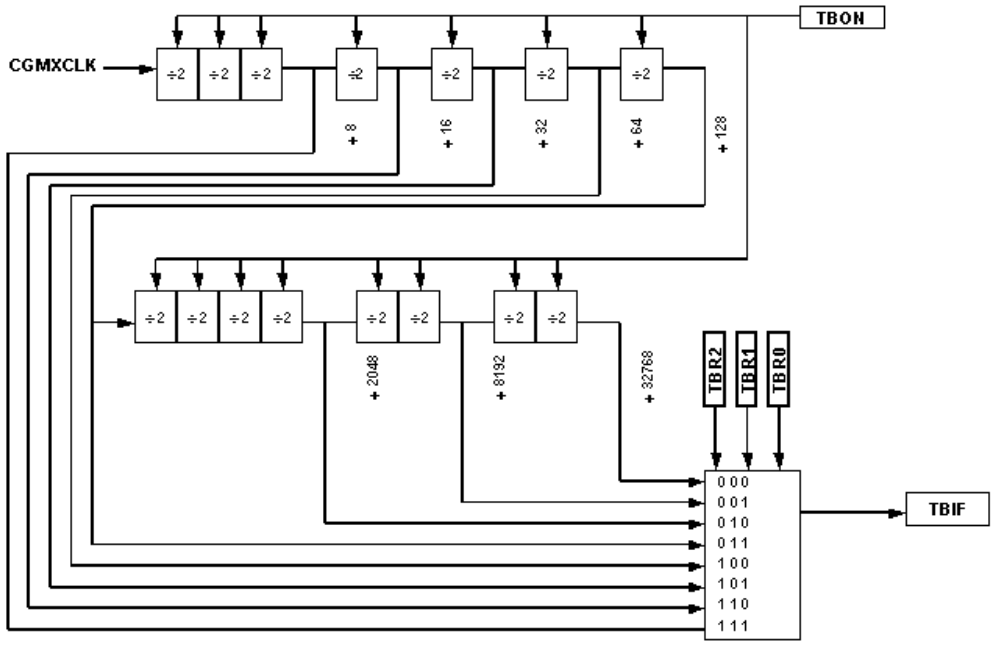
Selección de la velocidad de interrupción del TBM, para un oscilador de 32.768 kHz.

TBR2	TBR1	TBR0	DIVISOR	Velocidad Interrupción TB	
				Hz	ms
0	0	0	32.768	1	1000
0	0	1	8192	4	250
0	1	0	2048	16	62.5
0	1	1	128	256	-3.9
1	0	0	64	512	-2
1	0	1	32	1024	-1
1	1	0	16	2048	-0.5
1	1	1	8	4096	-0.24

Diagrama de Bloques del TBM

Ahora, se puede ver el funcionamiento del TBM a través del diagrama de bloques. El TBM genera una interrupción periódica dividiendo la frecuencia del cristal externo, CGMXCLK. El contador se compone de 15 etapas divisoras que se inicializan a 0 cuando se borra el bit TBON. Ocho de las 15 etapas son seleccionables por el usuario con los bits TBR2 a TBR0 de selección de velocidad del TBM. Estos bits permiten seleccionar una de las ocho velocidades de interrupción periódicas, como se vio en la tabla anterior.

El contador empieza contando cuando el bit TBON se pone a 1. Cuando el TBM ha contado hasta el valor seleccionado por TBR2-TBR0, el bit TBIF se pone a 1. Si el bit TBIE se pone a 1, se envía una petición de interrupción a la CPU. El indicador TBIF se borra escribiendo un 1 en el bit TACK. La primera interrupción que se genera después de activar el TBM ocurrirá aproximadamente a la mitad del periodo del desbordamiento. Los eventos subsecuentes ocurrirán en el periodo exacto.



Vectores de Interrupción del TBM

El TBM tiene su propio vector de interrupción para eliminar consultas por la fuente de interrupción. El mapa de los vectores mostrado es para el 68HC908GP32. Hay que verificar los datos técnicos de cada dispositivo para determinar la dirección de cada vector específico de cada 68HC08. El TBM tiene la prioridad más baja, así como su velocidad de interrupción periódica es relativamente más lenta.

FUENTE de INTERRUPCIÓN	Direcc. Vector	Flag	Máscara	Prioridad
TimeBase	$FFDC - $FFDD	TBIF	TBIE	16
Conversión Completa del ADC	$FFDE - $FFDF	COCO	AIEN	15
Pin del teclado KBD	$FFE0 - $FFE1	KEYF	IMASKK	14
SCI Transmisión Completa	$FFE2 - $FFE3	TC	TCIE	13
SCI Transmisión Vacía		SCTE	SCTIE	
SCI Entrada desocupada	$FFE4 - $FFE5	IDLE	ILIE	12
SCI Recepción completa		SCRF	SCRIE	
SCI Recepción excedida	$FFE6 - $FFE7	OR	ORIE	11
SCI Flag de Ruido		NF	NEIE	
SCI Error de cuadro		PE	FEIE	
SCI Error de Prioridad		PE	PEIE	
SPI Transmisión Vacía	$FFE8 - $FFE9	SPTE	SPTIE	10
SPI Recepción completa	$FFEA - $FFEB	SPFR	SPRIE	9
SPI Desbordamiento		OVRF	ERRIE	
SPI Modo Fallo		MODF	ERRIE	
TIM2 Desbordamiento	$FFEC - $FFED	TOF	TOIE	8
TIM1 Canal 1	$FFEE - $FFEF	CH1F	CH1IE	7
TIM2 Canal 0	$FFF0 - $FFF1	CH0F	CH0IE	6
TIM1 Desbordamiento	$FFF2 - $FFF3	TOF	TPOIE	5
TIM1 Canal 1	$FFF4 - $FFF5	CH1F	CH1IE	4
TIM1 Canal 0	$FFF6 - $ FFF7	CH0F	CH0IE	3
PLL	$FFF8 - $FFF9	PLLF	PLLIE	2
IRQ	$FFFA - $FFFB	IRQF	IMASK1	1
SWI	$FFFC - $FFFD	No	No	0
Reset	$FFFD - $FFFF	No	No	0

Ejercicio de Programación

Viendo las posibilidades del TBM, se puede escribir un programa para generar una señal periódica. Para este ejemplo, se quiere generar una señal cuadrada de 1 Hz, usando la salida del puerto C del 68HC908GP32. Se usa un cristal de 4.9152 MHz y se activan las interrupciones del TBM.

Además del programa principal, se escriben dos subrutinas. Se usa una subrutina para inicializar la MCU fuera de reset. Se usa una segunda subrutina como una rutina de servicio de interrupción. En la rutina de servicio de interrupción, se pone la interrupción del TBM, la conmutación (toggle) del puerto C y un reset en el contador del TBM.

```
* ---------------------------------------------------------------------
* Inicialización del 68HC908GP32
* ---------------------------------------------------------------------
       MOV     #%00001011,CONFIG1    ; Configura el registro CONFIG1
;                \\\\\\\\_____________ Desactiva el Módulo COP
;                 \\\\\\\_____________ Activa la instrucción STOP
;                  \\\\\\_____________ Recupera el modo Stop después de 4096 ciclos CGMXCLK
;                   \\\\\_____________ El LVI trabaja en modo 5V
;                    \\\\_____________  Activa el Módulo LVI
;                     \\\_____________  Activa el reset del módulo LVI
;                      \\_____________  Desactiva el LVI durante el modo STOP
;                       \_____________  Periodo de tiempo del COP de 262,128 ciclos

       MOV     #%00000011,CONFIG2    ; Configura el registro CONFIG2
;                \\\\\\\\___________ Usa el Reloj Interno del Bus como fuente para SCI
;                 \\\\\\\___________ Activa el Oscilador para operar durante el modo Stop
;                  \\\\\\___________ Regulador de Voltaje ON (Vdd > 3.6v)
;                   \\\\\___________ Sin implementar
       CLRA                          ;Inicializa el Acumulador
       LDHX    #ENDRAM               ;Stack Pointer -> Final de la RAM
       TXS                           ;       (H:X -> SP)
; FIN de la Inicialización de la MCU
```

La solución del ejercicio empieza con la subrutina de inicialización de la MCU. Se puede usar esta subrutina como una plantilla para todos los programas que se van a escribir.

Esta subrutina inicializa la MCU usando los registros CONFIG1 y CONFIG2. Estos registros sólo se pueden configurar una vez, después de un reset, para evitar cambiar la configuración de la MCU inadvertidamente durante una aplicación. Por lo que, es el primer paso que hay que hacer al empezar. En este

ejemplo, se usa el CONFIG1 para desactivar el COP para que no se tenga que alimentar el contador para evitar un reset. También se activa la instrucción STOP y se activa el LVI para permitir que trabaje a 5V.

En el registro CONFIG2, se selecciona el reloj del bus interno como fuente para la comunicación serie. También se permite trabajar al oscilador durante el modo STOP para que se pueda usar la característica del TBM de auto-despertarse periódicamente. La configuración del registro CONFIG2 no es crítica para esta aplicación. Alternativamente, se puede usar la inicialización predefinida para este ejemplo.

La siguiente instrucción, CLRA, inicializa el acumulador para borrar el registro de trabajo en el simulador/depurador, aunque esto no puede ser necesario en la aplicación. Las dos siguientes instrucciones remiten el puntero de pila al extremo de la RAM. Hay que recordar esto para guardar la compatibilidad con la familia HC05, la familia HC08 inicializa el puntero de pila automáticamente a $00FF después de un reset.

```
* ---------------------------------------------------------------------
* Aplicación
* ---------------------------------------------------------------------
        MOV     #$FF,DDRC               ; Activa el Puerto C como salida
        MOV     #%00000110,TBCR         ; Programa el registro Base de Tiempos
;                   \\\\\\\___________ Módulo Time Base ON
;                    \\\   \___________ Activa la Interrupción Base Tiempo
;                     \\\_____________ Selecciona Base Tiempos (Div 2^15)
;                                           @XTAL=4.9152MHz -> Frec=150Hz
        MOV     #COUNT,Counter          ; Inicializa el contador
        CLI                             ; Activa Interrupciones
        BRA     *                       ; Espera una interrupción
; FIN de la Aplicación
```

Esta sección de código es donde se empieza a trabajar con el TBM. Para generar una señal cuadrada de 1Hz, se tiene una conmutación de la salida cada 500ms. Para medir 500ms con un cristal de 4.9152MHz, se necesita usar el divisor de velocidad más alta de 215 o un valor de 32,768. Dividiendo la frecuencia por 32,768 da 150 eventos por segundo o 75 eventos para generar 500ms.

Para facilitar el proceso, se miden los eventos usando interrupciones. Se configura el puerto C como salida de la señal cuadrada. Después, se configura el registro TBM. Se activa la interrupción del TBM, se selecciona la velocidad del TBM por defecto y se conecta el Módulo. Se necesita un contador que cuente los 75 eventos antes de que conmute la salida. Se reserva un byte en la RAM para este contador y se inicializa usando un valor de 75. Finalmente, se activan las interrupciones con CLI y se espera para la interrupción de TBM.

```
TBM_Int_Serv:
        BSET    TACK,TBCR               ; Reconoce la Interrupción TBM
        DEC     Counter
        BNE     Exit                    ; Decrementa COUNTER y Exit
                                        ; si NO es 0
        COM     PORTC                   ; Si counter=0 ->Toggle Puerto C
                                        :   (tiempo = 0.5 segundos)
        MOV     #COUNT,Counter          ; Resets COUNTER
Exit:   RTI
; END TBM_Int_Serv
```

En la rutina de servicio de interrupción, se verifica si este tiempo para conmutar (‘toggle’) la salida. Para hacer esto, primero se reconoce la interrupción del TBM para evitar volver a entrar después de una RTI. Después, se decrementa el contador y se verifica si ha alcanzado un valor 0. Si no, se termina la subrutina y se espera para el próximo evento. Si el contador ha alcanzado un valor 0, esto significa que han pasado 500 ms. Entonces el puerto C conmuta, reinicia el contador a 75, y vuelve a empezar de nuevo.

Módulo de la Memoria Flash

Esta sección describe las características de la memoria Flash y cómo se programa. Se ven los pasos para borrar y programar la Flash y usar la Flash como un almacenamiento de datos no-volátil. También se describen los métodos diferentes para entrar en modo monitor, programando una parte en blanco y reprogramar una parte de la memoria.

Motorola ha usado varios tipos de tecnologías Flash en sus microcontroladores. Pero sólo se verá la última versión, llamada 'split gate' o Flash de segunda generación. Este tipo de Flash también se llama TSMC, que es uno de los sitios donde se fabrica la Flash. En general, la memoria Flash proporciona algunas ventajas muy importantes en cada aplicación, tanto en el diseño del producto, como en su uso y en el mantenimiento.

La Flash ofrece la posibilidad de programación en-circuito tanto en modo usuario, como en modo monitor. La programación en-circuito se está volviendo como el método preferido para programar dispositivos en producción y actualizar el código de la aplicación con un funcionamiento eficaz. Se puede realizar este tipo de trabajo incluso a través de un enlace telefónico remoto, un enlace inalámbrico RF o un enlace infrarrojo. La capacidad de programar y borrar un bloque pequeño, también permite reprogramar la Flash de una manera muy flexible. Esto permite borrar y reprogramar lo que se necesita, en lugar de tener que reprogramar la memoria entera. Usando este método, una aplicación puede continuar trabajando normalmente al mismo tiempo que el software se está poniéndose al día.

La Flash incluye una bomba de carga interna que elimina la necesidad de un alto voltaje separado para programarla. Esto reduce el costo en componentes y complejidad del circuito impreso. La Flash también incluye un mecanismo interno de seguridad que proporciona protección contra competidores que quieren copiar el software. La protección de bloque es otra forma de protección que ayuda a evitar la modificación del código inadvertidamente. Con la protección de bloque, se puede seleccionar el área de la memoria Flash a proteger. También se puede seleccionar para que se pueda saltar esta protección bajo ciertas configuraciones de hardware. Este característica protege contra un involuntario o malévolos cambio intentos o la destrucción del programa. También protege contra código ‘runaway’, que puede ejecutar el borrado o la reprogramación de la memoria Flash.

A continuación, se pueden ver las ventajas ofrecidas por la Flash 'split-gate'. La Flash 'split-gate' proporciona mejor tiempo de programación. Requiere entre 30 y 40 µs para programar cada byte. Esto significa que el tiempo de programación real para programar una memoria de 8 kbytes completa, está sobre un cuarto de segundo. Además, la Flash 'slit-gate' ofrece como mínimo 10,000 ciclos de programación/borrado en todo el rango completo de temperatura de trabajo. Esto es una gran mejora por encima de otras tecnologías que proporcionan aproximadamente 100 ciclos de programación/borrado a temperatura ambiente.

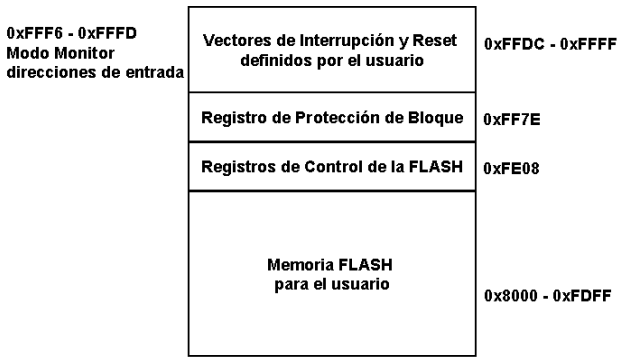
La Flash 'split-gate' usa un simplificado algoritmo de programación. Para programar una fila de bytes, se conecta el alto voltaje interno, se le aplica a la fila y entonces se escriben valores a cada byte de la fila. Esto difiere de la tecnología anterior, que requiere usar un proceso reiterativo de encender el alto voltaje y aplicarlo a una página, escribiendo valores a cada byte de la página, verificando todos los bytes para valores válidos en un margen de condición de lectura, repitiendo el proceso a todos los bytes a programar.

La Flash 'split-gate' incluye las opciones de protección de bloque más flexible y permite especificar un nivel más fino de granularidad, para la protección. La mayoría de dispositivos Flash 'split-gate' ofrecen 256 configuraciones de protección de bloque diferentes.

Programando un sólo byte, permite usar la memoria Flash para propósitos distintos de memoria de programa. Debido a la estructura de la Flash 'split-gate' y a la mayor duración, se puede asignar una porción de la memoria para el almacenamiento de datos no-volátiles y se puede eliminar la necesidad de colocar una EEprom fuera del circuito o una RAM con batería.

Mapa de Memoria de la Flash del 68HC908GP32

En el Mapa de Memoria se puede ver cómo está organizada la memoria Flash, se expresa en bytes de ocho bits. La memoria Flash siempre está en la parte alta de los 64k del mapa de memoria. Esto significa que una memoria de 8k va desde $E000 hasta $FFFF. La figura muestra la memoria de 32k del 68HC908GP32 en la que la Flash va desde $8000 hasta $FFFF. El monitor en ROM y algunas posiciones reservadas también residen en este rango, por lo que el número de posiciones disponibles de la Flash será aproximadamente de 500 bytes, menos que los 8k o 32k completos.



Hay una área reservada para una tabla de vectores en la dirección de la parte alta de la memoria Flash. El tamaño y el volumen de la tabla de vectores son específicos de cada dispositivo. Para determinar las posiciones exactas disponibles para la memoria de programa y la tabla de vectores en el dispositivo, hay que consultar el mapa de memoria de cada dispositivo del libro de datos técnicos.

La Flash 'split-gate' usualmente se programa una fila en un momento y se borra otra en una base de la página o se borra en masa. Una página siempre contiene dos filas de igual tamaño. El tamaño de la página es típicamente de 64 o 128 bytes, dependiendo del tamaño de la memoria Flash.

Los 68HC08 tienen una bomba de carga interna para programar y borrar la Flash, por lo que no se necesita una alimentación especial para este propósito. Se puede aplicar el voltaje generado internamente a la memoria para programar y borrar, poniendo a uno los bits del Registro FLCR de control de la Flash, que es un registro basado en RAM localizada en la parte alta de la memoria. Para programar y borrar la memoria Flash, secuencialmente se pone a uno el bit en FLCR y se escriben los valores en las posiciones designadas de la Flash. Este procedimiento requiere seguir unas pautas estrictas de tiempo. Después se verá el trabajo de borrado y programación en más detalle.

Registro FLCR de Control de la FLASH

El registro FLCR de control de la Flash, normalmente se localiza en la dirección $FE08 y consiste en cuatro bits que permiten programar o borrar la Flash.

FLCR	Bit 7	6	5	4	3	2	1	Bit 0
Leer:	0	0	0	0	HVEN	MASS	ERASE	PGM
Escribir:								
Reset:	0	0	0	0	0	0	0	0

- Cuando se activa el alto voltaje, el *bit HVEN* se pone a 1 y el alto voltaje se aplica a una página de la memoria durante un funcionamiento de borrado o de programación.
- Poniendo a 1 el *bit MASS* de control de borrado en masa, sirve para borrar la memoria entera.
- Como implica su nombre, poniendo a 1 el *bit ERASE*, se realiza una operación de borrado.
- Poniendo a 1 el *bit PGM*, se realiza una operación de programación.

Borrado de Página de la FLASH

1. Poner a 1 el bit ERASE y poner a 0 el bit MASS del registro FLCR.

2. Leer el registro FLBPR.

3. Escribir algún dato en alguna dirección de la FLASH dentro de la página para ser borrado.

4. Retardo para t_{nvs} (min 10 µs).

5. Activar el alto voltaje, poniendo a 1el bit HVEN del registro FLCR.

6. Retardo para t_{ERASE} (min 1 ms).

7. Poner a 0 el bit ERASE del registro FLCR.

8. Retardo para t_{nvh} (min 5 µs).

9. Apagar el alto voltaje, poniendo a 0 el bit HVEN del registro FLCR.

10. Retardo para t_{rcv} (min 1 µs), entonces la memoria puede ser accedida otra vez en modo lectura.

Primero, se pone a 1 el bit ERASE en el registro FLCR y se guardan los otros bits en el registro de borrado. Puesto que este registro se localiza en la memoria extendida, no se pueden usar las instrucciones bit-set (BSET) y bit-clear (BCLR) para alternar los bits del registro. En cambio, se deben usar las instrucciones LOAD y STORE. Seguidamente, se lee el registro de protección de bloque (FLBPR) y los datos guardados en cualquier dirección dentro de la página a ser borrada. Después de un retraso de tiempo de 10 µs (t_{nvs}), se pone a 1 el bit HVEN en el registro de control de la Flash. Se espera 1 ms (t_{erase}) y se pone a 0 el bit ERASE en el FLCR. Se espera otros 5 µs, (t_{nvh}) y se desconecta el alto voltaje, poniendo a 0 el bit HVEN. Cuando se vuelve de esta rutina, habrá pasado un tiempo t_{rcv} y se puede acceder a la Flash con la página borrada.

Se deben ajustar lo más posible los requisitos de tiempo en este procedimiento. Para verificar el valor de tiempo, hay que consultar la sección de características técnicas eléctricas del manual de usuario. Después de ejecutar este procedimiento, todos los bytes en esta página contendrán el valor de borrado $FF.

Borrado Total de la FLASH

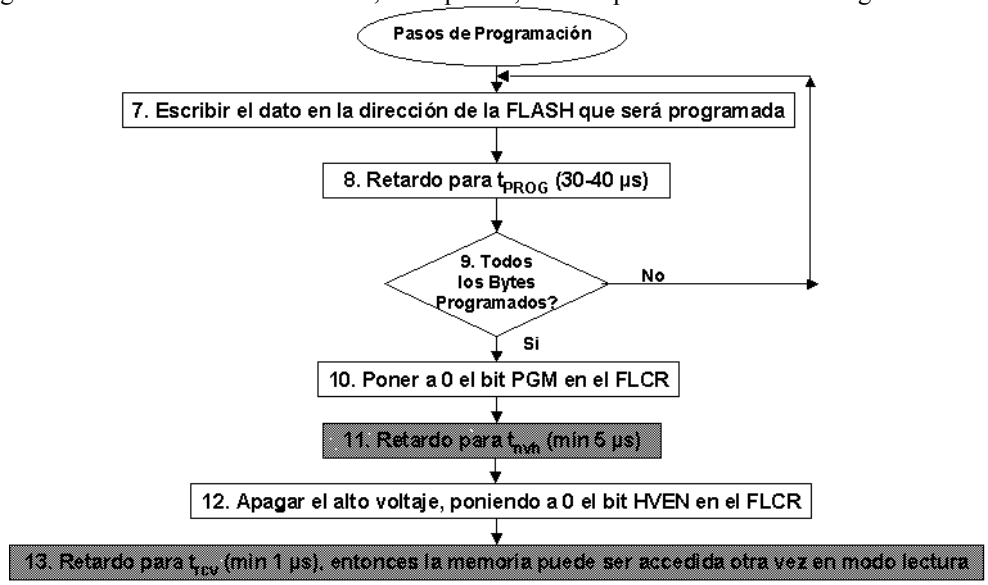
1. Poner a 1 el bit ERASE y el bit MASS del registro FLCR.
2. Leer el registro FLBPR.
3. Escribir algún dato en alguna dirección de la FLASH dentro del rango de direcciones de la FLASH.
4. Retardo para t_{nvs} (min 10 µs).
5. Activar el alto voltaje, poniendo a 1 el bit HVEN del registro FLCR.
6. Retardo para t_{MERASE} (min 4 ms).
7. Poner a 0 el bit ERASE del registro FLCR.
8. Retardo para t_{nvhl} (min 100 µs).
9. Apagar el alto voltaje, poniendo a 0 el bit HVEN del registro FLCR.
10. Retardo para t_{rcv} (min 1 µs), entonces la memoria puede ser accedida otra vez en modo lectura.

El procedimiento para borrar totalmente la memoria Flash, es similar al procedimiento del borrado de página con algunas diferencias. En el borrado en masa, se debe poner en el primer paso el bit MASS. En el paso tercero, se escribe en cualquier posición dentro de la memoria Flash. También, se debe alargar el retardo de tiempo en los pasos sexto y octavo. En el paso sexto, hay que utilizar un retardo de tiempo igual al tiempo de borrado en masa (t_{MERASE}). En el paso octavo, hay que utilizar un retardo de tiempo igual al tiempo que se necesita mantener el alto voltaje (t_{NVHL}).

Programación de la FLASH

1. Poner a 1 el bit PGM del registro FLCR.
2. Leer el registro FLBPR.
3. Escribir algún dato en alguna dirección de la FLASH dentro la columna a ser programada.
4. Retardo para t_{NVS} (min 10 µs).
5. Activar el alto voltaje, poniendo a 1 el bit HVEN del registro FLCR.
6. Retardo para t_{PGS} (min 5 µs).
Pasos de Programación

El procedimiento de programación de la Flash, se hace en el paso 13. Aquí se tienen los primeros seis pasos. Como se puede ver, el procedimiento de programación es similar al procedimiento de borrado. La diferencia principal es que el procedimiento de programación trabaja dentro de una fila de direcciones en lugar de una página o la memoria entera. También, en el paso 1, se debe poner el bit PGM en lugar del bit ERASE.



Otra diferencia entre este procedimiento y el procedimiento de borrado, es el lazo en los pasos del 7 al 9. En el lazo, se escribe un byte de datos en la dirección deseada, un retardo y entonces se verifica que todos los bytes se han programado. Este proceso se repite hasta que se termina de escribir a todas las posiciones que se quieren programar. Se debe prestar atención a los requisitos de tiempo, para todos los pasos de retardo en el procedimiento (véase los pasos 4, 6, 8, 11 y 13). Si no se permite suficiente tiempo durante cualquiera de estos pasos, se arriesga a programar las posiciones incorrectamente. Si los retardos son demasiado largos, se pueden perturbar posiciones vecinas programando bits erróneamente. Quizás una consideración más importante es que hay riesgo de acortar la vida de la memoria cuando esto ocurre.

Registro FBPR de Protección de Bloque de la FLASH

Ahora que ya se han visto los procedimientos para programar y borrar la Flash, se puede ver otro concepto importante, el llamado protección de bloque de la Flash. La protección de bloque permite proteger una memoria entera o una porción de ella, contra el borrado o el reprogramado. Esta protección se lleva a cabo usando el registro FLBPR de protección de bloque de la Flash, en la misma posición de la Flash. El valor en el registro FLBPR determina que cantidad de memoria se protegerá. Es decir, los bits del FLBPR especifican los ocho bits de una dirección de salida de 16 bits a ser protegidos.

FBPR

$FF7E	Bit 7	6	5	4	3	2	1	Bit 0
Leer: Escribir:	BPR7	BPR6	BPR5	BPR4	BPR3	BPR2	BPR1	BPR0
Reset:	U	U	U	U	U	U	U	U

Dirección de inicio de bloque de protección de la Flash:

1	Valor de FLBPR							0	0	0	0	0	0	0

Como se muestra en la figura, el primer bit de la dirección de salida se pone a 1 y los bits del 6 al 0 se ponen a 0. Con este mecanismo, la dirección de salida de protección de bloque, puede ser XX00 y XX80 (limitados a páginas de 128 bytes) dentro de la memoria Flash. La última dirección de protección de bloque siempre está al final de la memoria Flash, $FFFF.

Ejemplo de Dirección de Inicio

Se puede ver a continuación algunos ejemplos que usan el MC68HC908GP32. Si el FLBPR contiene un valor $00, entonces se protege la memoria entera. Si el registro contiene un valor $01, entonces el área protegida es $8080 - $FFFF. Si el registro contiene un valor $FE, entonces el área protegida es $FF0-$FFFF. Para el MC68HC908GP32 cada incremento del valor FLBPR, disminuye el área protegida a través de 128 bytes. La cantidad que disminuye por área protegida, depende del dispositivo. Hay que consultar los datos técnicos de protección específica para cada dispositivo. Cuando el FLBPR contiene un valor $FF, la memoria entera está indefensa. El registro FLBPR se borra y se programa de la misma manera, como cualquier otra posición de la Flash. La única manera de desconectar o cambiar el rango de protección del bloque, es aplicando el alto voltaje alrededor de VDD + 2.5v al pin externo IRQ. Cuando existe esta condición, se puede borrar el registro FLBPR. Esto libera el rango anteriormente protegido, haciéndolo disponible para borrar y reprogramar.

BPR[7:0]	Dirección de Inicio del Rango de Protección
$00	La memoria Flash entera es protegida
$01 (0000 0001)	$8080 (1000 0000 1000 0000)
$02 (0000 0010)	$8100 (1000 0001 0000 0000)
y así sucesivamente...	
$FE (1111 1110)	$FF00 (1111 1111 0000 0000)
$FF	La memoria Flash entera NO es protegida

Nota: la dirección final del rango es siempre $FFFF.

Características del Monitor del 68HC08

El modo monitor está disponible en todos los dispositivos 68HC08. El modo monitor se usa principalmente para proporcionar el acceso serie al dispositivo y a sus módulos internos para programarlos, leerlos, ponerlos a punto y probar sus actividades. Se debe usar este modo para programar un dispositivo en blanco. Esto se hace a través de un sólo pin del puerto, reteniendo toda la otra funcionalidad del pin en modo usuario.

El monitor incluye seis potentes comandos que permiten llevar a cabo algunas características importantes, que incluyen descargar un programa en la RAM e iniciar la ejecución del programa y la interface con el dispositivo, para realizar una programación en circuito o fuera del circuito. El modo monitor también incluye un esquema de seguridad que reduce la posibilidad de acceso desautorizado a su código.

Comandos en Modo Monitor

La mayoría de las funciones del modo monitor típicamente se realizan transmitiendo un programa en la RAM y ejecutándolo. Esto se hace a través de seis comandos del monitor.

Comando	Descripción	Opcode	Dato devuelto	Operando
READ	Lee el byte desde la memoria	$4A	Devuelve el contenido de la dirección especificada	Dirección 2 bytes en orden: byte alto: byte bajo
WRITE	Escribe el byte a la memoria	$49	Ninguno	Dirección 2 bytes en orden: byte alto:byte bajo; byte bajo seguido por byte dato
IREAD	Escribe a la última dirección accedida +1	$1A	Devuele los 2 bytes siguientes en la memoria desde las últimas direcciones acced	Dirección 2 bytes en orden byte alto:byte bajo
IWRITE	Lee el Stack Pointer	$19	Ninguno	Simple byte de dato
READSP	Ejecuta las instrucciones	$0C	Devuelve la dirección de 2 bytes del Stack Pointer en orden: byte alto:byte bajo	Ninguno
RUN	Ejecuta las instrucciones PULH y RTI	$28	Ejecuta las instrucciones PULH y RTI	Ninguno

- Para leer una dirección en particular, se entra la orden *READ* seguido por una dirección. El monitor devuelve el valor guardado en esa dirección.
- Para escribir en una dirección en particular, se entra la orden *WRITE* seguido por la dirección que se quiere escribir y los datos que se quieren escribir.
- La orden *IREAD* no requiere ningún operando y devuelve los siguientes dos bytes de la última dirección accedida.
- La orden *IWRITE* escribe un solo byte de dato en la última dirección que se accedió incrementada en uno.

• La orden READSP no requiere operandos y devuelve los dos bytes del puntero de pila (stack pointer).

• La orden *RUN* no requiere ningún operando y ejecuta las instrucciones PULH y RTI. Ejecutando este juego de instrucciones, el contador de programa se apila en una nueva posición. Por consiguiente, se deben poner los valores de las posiciones en la pila antes de usar la orden RUN. Poniendo estos valores eficazmente, se pondrá el contexto cuando se ejecute la instrucción RTI.

Se debe consultar el manual de usuario para la información sobre el tiempo apropiado para comunicar con el modo monitor.

Programación de la FLASH

•Programación de una parte de la memoria en blanco

- Modo Monitor fuera del circuito
- Modo Monitor en-circuito con entrada monitor estándar
- Modo en-circuito con entrada monitor forzada

• Reprogramación de una parte de la memoria

- Modo Usuario
- Modo Usuario y modo monitor
- Modo Monitor con entrada monitor estándar

Antes se describió cómo se debe usar el modo monitor para programar una parte de la memoria en blanco. Hay dos métodos disponibles para hacerlo: Programar la memoria fuera del circuito y Programar la memoria en circuito. La programación fuera del circuito es el método más fácil de usar y el más antiguo. Sin embargo, si se quiere más control de la programación se debe usar la programación en circuito. Al usar la programación en circuito, se deben satisfacer los requisitos de entrada en modo estándar, que usa cualquier entrada monitor estándar o entrada monitor forzada. La entrada monitor forzada es una nueva forma, es una alternativa a la entrada monitor estándar.

Una vez se ha programado la Flash, hay tres métodos que se pueden usar para reprogramar una parte de la memoria Flash. Se describirá en detalle cada uno de estos procedimientos y se empezará con el método de programar una parte en blanco.

Herramientas de Programación Fuera del Circuito

El método más fácil para programar una parte de la memoria en blanco, es la programación fuera del circuito. Con este método, no se tienen que seguir los procedimientos de programación o satisfacer los requisitos de entrada en modo monitor. El programador se cuida de estos detalles.

Para programar fuera del circuito, simplemente hay que conectar el dispositivo al programador, cargar el archivo S-record y después pulsar un botón. El dispositivo se programará en unos segundos. Motorola proporciona simuladores en circuito para muchos dispositivos 68HC08 que permiten la programación fuera del circuito.

Para otros dispositivos, la programación se hace con el equipo SPGMR08, una placa adaptadora apropiada y el software MCUscribe. Se proporciona mayor información sobre las herramientas de desarrollo para el 68HC08 en la web (http://www.mcu.motsps.com/documentation/68HC08/devhc08.html) y en particular, la Guía de Selección de Herramientas de Desarrollo para Microcontroladores, que describe las opciones que están disponibles para programar los dispositivos. También muestra soluciones de programación de Motorola y de terceros fabricantes.

Entrada en Modo Monitor

La programación en circuito es el método preferido para programar una parte de la memoria en blanco. Aunque el monitor es un dispositivo muy simple, se debe satisfacer toda condición de entrada para accederlo. Estas condiciones están satisfechas poniendo los pins a los voltajes específicos durante el 'reset'. Las dos maneras de entrar en modo monitor son la entrada monitor estándar y la entrada monitor forzada. Todos los dispositivos 68HC08 soportan la entrada monitor estándar. Usando este método, el dispositivo puede estar gobernado a cualquier frecuencia dentro del rango permitido en modo usuario. Sin embargo, para facilitar la comunicación entre el dispositivo y el 'host', el reloj debe seleccionarse para lograr un 'baud rate' determinado. En la mayoría de dispositivos 68HC08, el 'baud rate' es igual a la frecuencia de trabajo interna dividida por 256. Una frecuencia de reloj externa típica es de 4.9152 MHz que produce un 'baud rate' común de 9600 bits por segundo.

La mayoría de los dispositivos 68HC08 más nuevos soportan un método simplificado para entrar en el modo monitor llamado 'entrada monitor forzada'. Debido a su concepto de interface simplificada, este método soporta lo que se ha denominado programación "FLASHwire". Es una característica de seguridad que proporciona protección contra un acceso desautorizado para programar la memoria. Durante la entrada en modo monitor, el dispositivo 'host' requiere un código de seguridad para entrar en forma de ocho bytes de datos. El valor entrado se compara con el valor en la posición $FFF-$FFFD. Si los valores se emparejan, se puede leer,

borrar o programar la memoria. Si los valores no se emparejan, se prohíbe el acceso para programar la memoria. En este caso, la operación que sólo se puede realizar en la Flash del 68HC08, es un borrado completo de la memoria Flash. Todos los otros módulos del dispositivo, incluso la RAM, están disponibles para su inspección y usarlos sin tener en cuenta el acceso de seguridad de la Flash. Se verán algunos ejemplos específicos de programación de una parte de la memoria en blanco.

Circuito Modo Monitor del 908GP32

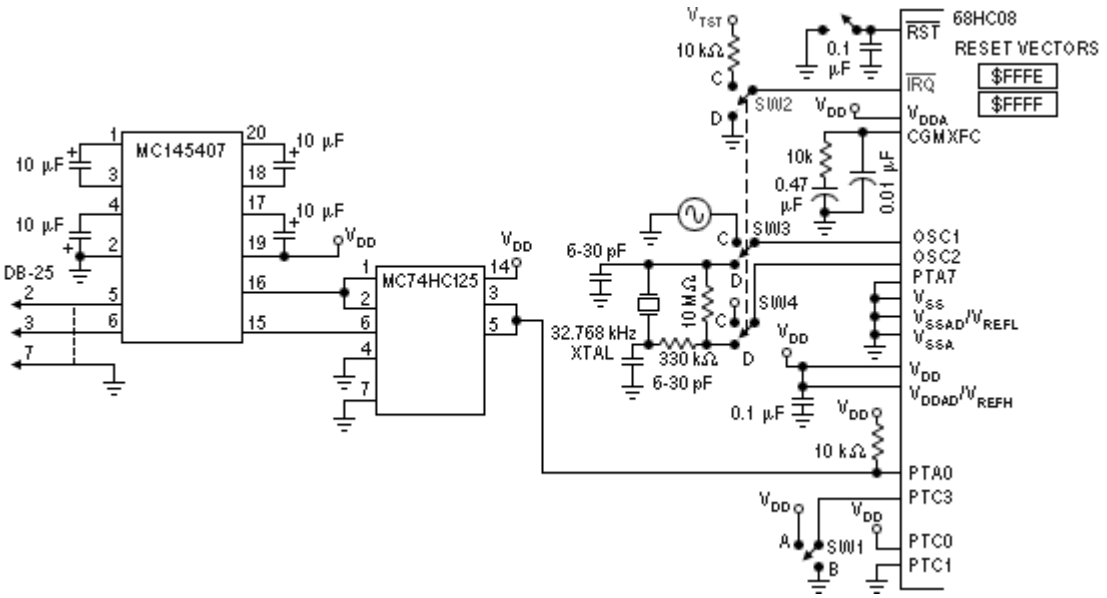
Se puede ver como ejemplo, la entrada en modo monitor estándar utilizando el circuito 68HC908GP32. Estos requisitos generalmente son iguales para todos los dispositivos 68HC08. Sin embargo, se deben verificar los datos técnicos de cada dispositivo para determinar la configuración de cada pin para entrar en modo monitor.

Para poner el 68HC908GP32 en modo monitor, hay que poner el pin PTC0 a V_{DD} y el pin PTC1 a tierra. El pin PTC3 especifica si la frecuencia interna es la frecuencia del oscilador dividida por 2 o dividida por 4. Para poner el pin PTC3 a 1, SW1 se pone en la posición A para dividir por 4 y se pone en la posición B es para dividir por 2.

Para simplificar la interface en modo monitor, esta opción no se ofrece en muchos dispositivos 68HC08. La frecuencia interna siempre es la frecuencia externa dividida por 4 como en modo usuario.

Otra condición para la entrada en modo monitor estándar es el voltaje Vtst en el pin IRQ. En algunos dispositivos se puede ver que este voltaje se le llama $V_{DD}+V_{HI}$. SW2 se debe poner a la posición C para aplicar V_{TST} a IRQ para la entrada monitor estándar. SW2 se puede poner en la posición D si se está usando un cristal externo y el PLL interno con una parte de la memoria en blanco. Para mayor información sobre el rango de nivel de voltaje para cada dispositivo, se puede consultar en el libro de datos técnicos del dispositivo.

Cuando se pone SW2 en la posición C, también se debe poner SW3 y SW4 en la posición C. Por eso, para la entrada en monitor estándar, se debe conectar un oscilador externo a OSC1 y V_{TST} aplicado al pin IRQ. Para la entrada en modo monitor forzada, la posición D para SW2, SW3 y SW4 representan el caso donde se conecta OSC1 y OSC2 a un cristal de 32.768 KHz, con IRQ conectado a masa para activar el PLL interno en una parte de la memoria en blanco.



Seguidamente se puede ver el pin de comunicación. Para el 68HC908GP32, este pin es el PTA0. Se necesita usar una resistencia de 'pull-up' en el pin de comunicación. Esto mantendrá el estado inactivo de la línea de comunicación en el nivel apropiado.

El modo monitor usa la comunicación bidireccional, half duplex y de no retorno a cero. Esto requiere multiplexar la línea de recepción/transmisión del 'host', que se puede implementar fácilmente con un buffer de tres estados (MC74HC125).

Otro componente, es el convertidor de niveles de tensión entre RS-232 y los niveles V_{DD}/V_{SS}. Para reducir componentes y el coste de fabricación, se puede usar una placa intermedia para realizar el circuito de multiplexado y el convertidor de niveles.

En el 68HC908GP32, cuando el nivel del pin PTA7 pasa a 0, con estas señales, entrará el código de seguridad al dispositivo de forma serie sobre el pin PTA0.

Un método alternativo es usar el puerto A de 8-bits y entrar el primer código byte en un momento. En algunos dispositivos, esta funcionalidad se pone en un pin diferente. En otros modelos 68HC08, esta funcionalidad se consolida en pin PTA0. Por ejemplo, el 68HC908JL3 no usa el pin PTA7 para la entrada de señales de código de seguridad vía serie.

Entrada en modo Monitor Forzada

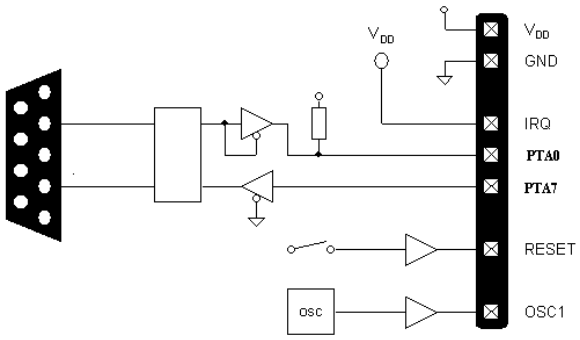
La mayoría de los nuevos 68HC908 soportan la entrada de modo monitor forzada. Con este método de entrada, entra automáticamente en el modo monitor cuando parte de la memoria está en blanco, indicado por un vector de reset en blanco. Esto elimina la necesidad de aplicar un voltaje en IRQ y la necesidad de satisfacer condiciones del puerto para la entrada en modo monitor. Hay que tener presente que se puede necesitar un pin adicional para hacer que el monitor trabaje como quiera. Para mayor información, hay que consultar el libro de datos técnicos de cada dispositivo.

Por ejemplo, en algunos dispositivos como el 68HC908JL3, no se requiere ninguna configuración del puerto adicional para el funcionamiento apropiado del modo monitor. Una vez en modo monitor, sólo se necesita enviar el código de seguridad en el pin de comunicación (puerto B0). En otros dispositivos, como el 68HC908GP32, algunos pins son verificados después de entrar en modo monitor, por lo que se necesita configurar éstos pins apropiadamente. En el 68HC908GP32, se puede activar o se puede desviar la inicialización del PLL poniendo el pin IRQ a V_{SS} o V_{DD} respectivamente. También, se debe poner el pin PTA7 a V_{SS} para indicar que el código de seguridad se transmitirá sobre el puerto de comunicación serie (pin PTA0).

En dispositivos que no soportan la entrada en modo monitor forzada, como el 908AZ60 y el 908RK2, se debe usar la entrada modo monitor estándar para programar una parte de la memoria en blanco. Para estos dispositivos, se debe aplicar un voltaje alto en IRQ y las polaridades específicas en varios pins del puerto. Estos requisitos son los mismos que para reprogramar una parte que usa la entrada en modo monitor estándar. Más adelante, se describirá la reprogramación en más detalle.

Interface ISP para el GP32

La figura siguiente muestra el modo monitor forzado, la interface para la programación en circuito para el 68HC908GP32. Al usar la entrada en modo monitor forzada, las únicas líneas que deben ir a la placa de circuito impreso son la línea de comunicación y la referencia a masa. Dependiendo de que tipo de programador 'host' se esté usando, todas las otras líneas se pueden controlar en la placa de aplicación y no se necesitan para esta. Se pueden controlar las líneas usando 'jumpers', interruptores digitales o a través de un cableado fijo a una polaridad apropiada. Se pueden ver cada una de estas líneas en más detalle.



Cuando el vector de reset está en blanco, se puede entrar en modo monitor aplicando V_{TST} o V_{DD} a IRQ. Si se aplica V_{TST}, se puede requerir otra configuración del puerto para entrar en modo monitor. Si V_{DD} se aplica a IRQ, no se requiere ninguna otra configuración del puerto.

La frecuencia interna es la frecuencia externa (oscilador) dividida por 4 y la velocidad de comunicación es la frecuencia interna dividida por 256. La fuente de reloj en OSC1, puede residir en la placa de circuito impreso o puede alimentarse como parte de la interface de programación.

En la familia GP y GR, se tiene la opción de aplicar un tercer voltaje en IRQ. En estos dispositivos, la parte puede ser gobernada por un cristal de baja frecuencia, típicamente de 32 KHz y un PLL interno. V_{SS} se puede aplicar a IRQ cuando el vector de reset está en blanco, en caso que el PLL está automáticamente acoplado. El PLL se pondrá la frecuencia interna a 2.4576 MHz cuando el reloj externo (cristal o oscilador) sea de 32.768 KHz. En este caso, la velocidad de comunicación es de 9600 bits por segundo y no se requiere ninguna otra configuración del puerto.

La habilidad de utilizar un reloj con un cristal de baja frecuencia y después aumentar la frecuencia hasta un nivel para producir un 'baud rate' común, sólo está disponible en una memoria en blanco. También, si el PLL está acoplado, entonces los componentes externos de filtrado necesitan estar conectados al pin CGMXFC (véase los esquemáticos del modo monitor del 68HCGP32). Nota de aplicación AN1770, Programación en circuito de la memoria Flash del 68HC908GP20, describe en detalle esta y otras condiciones de entrada en modo monitor.

El puerto PTA0 de comunicación conectado a través de la interface de programación y se puede unir al circuito convertidor/multiplexor de nivel, localizado dentro o fuera de la placa de producción para reducir su coste. También, se puede incluir este circuito en una ensamblaje intermedio fuera de la placa de producción para conectar el 'host' a la placa.

Los componentes de interface no se necesitan cuando el 'host' no comunica sobre un enlace RS-232, sino que usa el mismo nivel de voltaje que PTA0. Éste es el caso donde el módulo de programación usa otro 68HC908GP32 para comunicar más de uno de sus puertos bi-direccionales. Cualquiera que sea la configuración, es importante recordar que se debe conectar una resistencia de 'pull-up' al puerto de comunicación, para una comunicación fiable en modo monitor.

Se necesita PTA7 para especificar la entrada serie del código de seguridad. Esta configuración no se necesita en algunos dispositivos como el 68HC908JL3. La línea de 'reset' sólo se necesita si el programador del 'host' lo requiere. En la mayoría de casos, con tal de que el dispositivo se ponga inicialmente en modo monitor, el 'host' puede programar la memoria sin la línea de 'reset'.

Herramientas de Programación en circuito

A continuación se describen las herramientas que pueden servir como un 'host' de programación. En la mayoría de las situaciones, se pueden usar las mismas herramientas que se usan para la programación fuera del circuito, que para la programación en circuito. Los simuladores en circuito (ICS) están disponibles para los diferentes dispositivos 68HC08 y sirven para programar estos dispositivos en la placa del usuario.

Todos éstos sistemas ICS, como mínimo, soporta el interface de entrada en modo monitor estándar, llamado "mon08". Se puede implementar esta interface a través de un conector y las conexiones necesarias en la placa de usuario. Los simuladores ICS esencialmente interceptan las conexiones del modo usuario normal en el pin del dispositivo en modo monitor y aplicar las señales necesarias para entrar en modo monitor y programar el dispositivo. Este tipo de interconexión previene la posibilidad de contención de la señal en la placa usuario.

Cuando la ICS no se conecta al conector "mon08" de la placa de usuario, pone los 'jumpers' en las dos filas del conector para activar el flujo de la señal durante el funcionamiento normal. Algunos de los nuevos sistemas ICS, como el ICS08JLJK para el JL3 y JK3, han incorporado una interconexión de pins mínimo para la programación de parte en blanco.

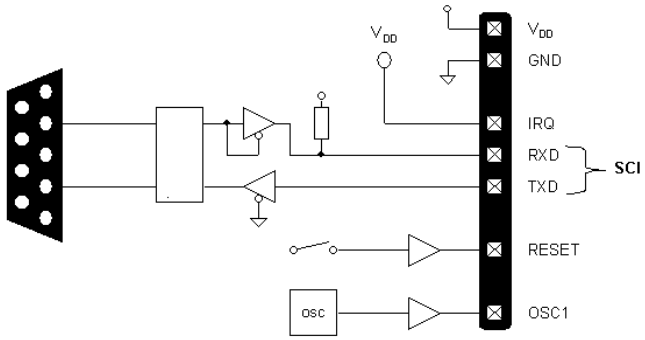
Esta interface consiste en dos hilos, la línea de comunicación y la conexión a masa y depende del hardware del usuario, para asegurar que no hay ninguna contención de señal en la línea de comunicación. Esta ICS también ofrece una salida de reloj en caso de que se necesite en la placa usuario.

Algunos dispositivos, como el 68HC908AZ60 no tienen una placa ICS. Una opción que se tiene en este circuito para programar, es usar el programador SPGMR08 como hardware intermedio entre el dispositivo objetivo y un PC con el MCUscribe. El SPGMR08 mantienen la conversión de niveles necesaria y el multiplexando de la línea de comunicación. Adicionalmente, proporciona una fuente de voltaje alto y un reloj conveniente para un 'baud rate' común. Se puede acceder a todas estas señales a través de uno de los conectores de la placa de adaptación en el SPGMR08 y se pueden traer a la placa del usuario. Estas conexiones se reproducen esencialmente en el entorno de programación fuera del circuito que usa el SPGMR08 y una placa adaptadora de programación.

En algunas situaciones, puede ser más conveniente escribir su propio programa software en el 'host'. Por ejemplo, quizás se quiere usar un entorno 'host' o tener únicas circunstancias de programación que no son soportadas para el software de programación disponible. En estos casos, el programa 'host' se debe adherir cerca al protocolo de comunicación de modo monitor y velocidad de bit. La práctica normal es para el 'host' para descargar una rutina a la RAM del dispositivo. Esta rutina de la RAM contendrá las rutinas de programación y una comunicación ejecutiva para recibir órdenes y datos desde el 'host' para programar la Flash. La nota de aplicación AN1770 describe en detalle este programa ‘host’.

Reprogramación en Modo Usuario

El siguiente método para reprogramar un dispositivo en circuito involucra la realización de operaciones en modo normal y en modo monitor. Para los dispositivos que soportan la entrada en modo monitor forzada, es mejor primero borrar el micro o aparece en blanco en modo usuario. Después, se puede usar la entrada en modo monitor forzada para programar el micro. Como se describió antes, la entrada en modo monitor forzada, elimina la necesidad de los requisitos de entrada en modo monitor estándar, incluyendo V_{TST} en IRQ. Para emplear la entrada en modo monitor forzada, el vector de reset ($FFFE-$FFFF) debe estar en blanco. En modo usuario, primero se realiza un borrado de página en la fila del vector o una operación de borrado en masa y después hacer un ‘reset’ del dispositivo. El dispositivo entrará en modo monitor, usando la entrada en modo monitor automáticamente. Una vez en modo monitor, se pueden realizar todas las operaciones de programación usando el puerto de comunicación del monitor. Aunque este método usa la entrada en modo monitor forzada, de este modo se reducen los requisitos de entrada en modo monitor, usando este sistema se requiere satisfacer algunas condiciones hardware.

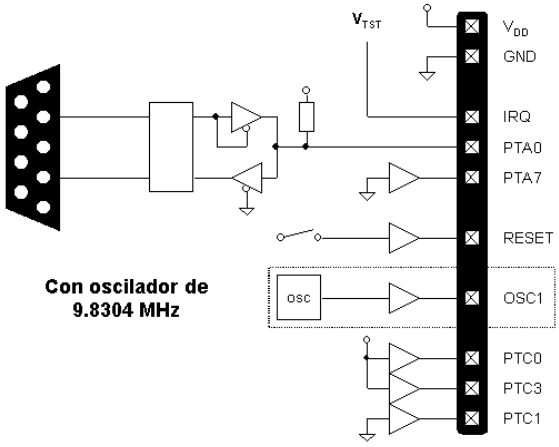


Dado esto, hay en algunas circunstancias que hacen que este sea el método preferido para reprogramar un micro en circuito. Por ejemplo, a menudo ya se han establecido conexiones para la comunicación de un pin de un puerto, porque el dispositivo se programó en un circuito inicialmente.

Otro ejemplo sería que cuando el dispositivo no tiene un SCI y el tamaño de la RAM es demasiado pequeña para mantener el código de comunicación de 'bit-bang' y las rutinas de programación necesarias para permitir la programación en modo usuario.

Reprogramación con entrada monitor estándar

El tercer método para reprogramar un micro en circuito es realizar todas las operaciones de programación en modo monitor, usando la entrada monitor estándar. En entrada monitor estándar, si el vector de reset no está en blanco, entonces se debe aplicar V_{TST} a IRQ. Además, la entrada en modo monitor estándar requiere satisfacer ciertas condiciones en modo de entrada, respecto a las líneas utilizadas, que las condiciones de entrada varían según la combinación de líneas y se pueden incluir seleccionando la proporción de división para la frecuencia de trabajo y poniendo un pin a masa, normalmente el pin PTA7, para seleccionar la entrada serie para el código de seguridad.



También, esta entrada en modo estándar no emplea el PLL automáticamente. Esto puede ser preocupante en aplicaciones donde el reloj de la placa es un cristal de 32.768 KHz. En este caso, se necesita configurar un reloj externo para manejar el dispositivo o las comunicaciones empezarán a una velocidad muy lenta, alrededor de 32 o 64 baudios. Esto le exige al programador del 'host' que tenga una velocidad de comunicación lenta. Por consiguiente, se recomienda que inicie el dispositivo PLL lo más pronto posible para activar una velocidad de comunicación más rápida. Haciendo una comparación con este método y el segundo método que se ha descrito anteriormente, usando una combinación de modo usuario y modo monitor para reprogramar un micro. En el segundo método, no se necesita usar un reloj externo cuando el reloj de la placa es de 32.768 KHz. Esto es porque con la entrada de modo monitor forzada, el IRQ se puede manejar a un nivel de V_{SS}. En este caso, el PLL se inicia automáticamente y produce una velocidad de comunicación de 9600 baudios.

Resumen de Métodos de Reprogramación

Método 1 - Modo Usuario

- Método preferido para la reprogramación de un micro en circuito.
- Proporciona comunicación asíncrona eficaz y robusta.
- Puede ser usado con o sin SCI.

Método 2 - Modo Usuario y Monitor (entrada monitor forzada)

- Usa la entrada en modo monitor forzada.
- Puede ser usada cuando el micro no incluye SCI y la RAM es pequeña.

Método 3 - Modo Monitor (entrada monitor estándar)

- Necesita satisfacer todos los requerimientos de entrada.
- Si el vector de reset no está en blanco, se puede aplicar V_{TST} a IRQ.
- No emplea automáticamente el PLL.

Programación de la Flash basada en la ROM interna

Para dispositivos que tienen una cantidad pequeña de RAM, se ha instalado unas rutinas de programación en la ROM interna accesible por el usuario. Estas rutinas se han instalado en unos dispositivos 68HC08 porque el espacio de la RAM es demasiado pequeña para soportar la rutina de programación, una rutina de comunicación, un buffer de datos de algún tamaño significante, la pila y las variables de la aplicación.

Actualmente, los dispositivos 68HC908 que incluyen estas rutinas internas son el GR8, KX8, JL3, JK3, JK1, y el JB8. Las rutinas basadas en la ROM interna, incluyen una rutina para programar un rango, una rutina para borrar una página o la memoria entera, una rutina para leer y verificar un rango de la Flash, una rutina para recibir un byte de datos en el puerto de comunicación en modo monitor y una rutina de retardo que mantiene el tiempo correcto durante las operaciones de borrado/programación de la Flash.

Estas rutinas ayudan a minimizar el tiempo de desarrollo y escribir rutinas a la RAM compactas para programar y borrar la Flash. Las rutinas soportan operaciones de la Flash en una amplia gama frecuencias de trabajo, mientras que mantiene conformidad a las especificaciones de programación. Si se está usando uno de los dispositivos que no incluyen estas rutinas basadas en la ROM interna, sirven como una buena referencia para escribir sus propios programas de la Flash. Para dispositivos que tienen una RAM más grande, como el 68HC908GP32, se puede copiar esta rutina en la rutina de usuario de la RAM.

La nota de aplicación AN1831 describe cómo usar estas rutinas e incluye ejemplos, diagrama de flujo de la rutina y código fuente.

Nuevas instrucciones del 68HC08

El conjunto de instrucciones de la familia 68HC08 es una extensión del conjunto de instrucciones de la familia 68HC05. Esta sección contiene únicamente las instrucciones de la familia 68HC08.

AIS - Suma el valor Inmediato (con signo) al Puntero de Pila
AIX - Suma el Valor Inmediato (con signo) al Registro de índice
BGE - Bifurcación si es Mayor que o Igual a (operandos con signo)
BGT - Bifurcación si es Mayor que (operandos con signo)
BLE - Bifurcación si es Menor que o Igual a (Operandos con signo)
BLT - Bifurcación si es Menor que (Operandos con signo)
CBEQ - Compara y Bifurca si es Igual
CBEQA - Compara A con Inmediato, Bifurca si es Igual
CBEQX - Compara X con Inmediato, Bifurca si es Igual
CLRH - Borra la parte alta del Registro de Índice (H)
CPHX - Compara el Registro de Índice con la Memoria
DAA - Ajuste Decimal del Acumulador
DBNZ - Decrementa y Bifurca si no es Cero
DIV - Divide
LDHX - Carga el Registro de Índice con la Memoria
MOV – Mueve
NSA - Cambia los “nibbles” del Acumulador
PSHA - Pone el Acumulador en la Pila
PSHH - Pone la parte alta del Registro Índice (H) en la Pila
PSHX - Pone la parte baja del Registro Índice (X) en la Pila
PULA - Saca el Acumulador de la Pila
PULX - Saca la parte baja del Registro Índice (H) de la Pila
PULH - Saca la parte alta del Registro Índice (H) de la Pila
STHX - Guarda el Registro de Índice
TAP - Transfiere el Acumulador al Registro de Código de Condición
TPA - Transfiere el Registro de Código de Condición al Acumulador
TSX - Transfiere el Puntero de Pila al Registro de Índice
TXS - Transfiere el Registro de Índice al Puntero de Pila

AIS - Suma el valor Inmediato al Puntero de Pila (con signo)

Operación: **SP ← (SP) + (16 « M)**

Descripción: Suma el operando inmediato al Stack pointer (SP). El valor inmediato es un operando de 8 bits complemento a dos con signo. El operando de 8 bits es extendido a 16 bits con signo, anterior a la suma. La instrucción AIS se puede usar para crear y quitar un 'buffer' de la zona del stack, que se usa para guardar temporalmente las variables.

Esta instrucción no afecta ningún bit de código de condición, para que la información de estado se pueda pasar a/o de una subrutina o función C y asignando o no el espacio para las variables locales que no perturbarán esa información de estado.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
AIS #opr	IMM	A7	ii ii	2

AIX - Suma el Valor Inmediato al Registro de índice (con signo)

Operación: **H:X ← (H:X) + (16 « M)**

Descripción: Suma el operando inmediato al Registro de índice de 16 bits, formado por la concatenación del registro H y X. El operando inmediato es un offset de 8 bits complemento a dos con signo. El operando de 8 bits es extendido a 16 bits con signo anterior a la suma.

Esta instrucción no afecta ningún bit de código de condición, para que los cálculos del puntero del registro de índice no perturbará el código circundante que puede consultar de forma segura el estado de los bits de estado del CCR..

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
AIX #opr	IMM	AF	ii	2

BGE - Bifurcación si es Mayor que o Igual a (operandos con signo)

Operación: **PC ← (PC) + $0002 + *rel*** **si (N ⊕ V) = 0**

Por ejemplo, si (A) (M) (números complemento a dos con signo)

Descripción: Si la instrucción BGE se ejecuta inmediatamente después de la ejecución de una instrucción de comparación o de substracción, la bifurcación ocurre si y sólo si, el número complemento a dos representado por el registro interno apropiado (A, X o H:X) era mayor que o igual, al número complemento a dos representado por M.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
BGE opr	REL	90	rr	3

BGT - Bifurcación si es Mayor que (operandos con signo)

Operación: **PC ← (PC) + $0002 + *rel*** **si Z | (N ⊕ V) = 0**

Por ejemplo, si (A) > (M) (números complemento a dos con signo)

Descripción: Si la instrucción BGT se ejecuta inmediatamente después de la ejecución de una instrucción CMP, CPX, CPHX o SUB, la bifurcación ocurre si y sólo si, el número complemento a dos representado por el registro interno apropiado (A, X o H:X) era mayor que el número complemento a dos, representado por M.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
BGT opr	REL	92	rr	3

BLE - Bifurcación si es Menor que o Igual a (operandos con signo)

Operación: **PC ← (PC) + $0002 + *rel*** **si Z | (N ⊕ V) = 1**

Por ejemplo, si (A) (M) (números complemento a dos con signo)

Descripción: Si la instrucción BLE se ejecuta inmediatamente después de la ejecución de una instrucción CMP, CPX, CPHX o SUB, la bifurcación ocurre si y sólo si, el número complemento a dos representado por el registro interno apropiado (A, X o H:X) era menor que o igual al número complemento a dos representado por M.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
BLE opr	REL	93	rr	3

BLT - Bifurcación si es Menor que (Operandos con signo)

Operación: **PC ← (PC) + $0002 + *rel*** **si Z (N ⊕ V) = 1**

Por ejemplo, si (A) < (M) (números complemento a dos con signo)

Descripción: Si la instrucción BLT se ejecuta inmediatamente después de la ejecución de una instrucción CMP, CPX, CPHX o SUB, la bifurcación ocurrirá si y sólo si, el número complemento a dos representado por el registro interno apropiado (A, X o H:X) era menor que o igual al número complemento a dos representado por M.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
BLT opr	REL	91	rr	3

CBEQ - Compara y Bifurca si es Igual

Operación: **(A) – (M);**
PC ← (PC) + $0003 + rel **si el resultado es $00**
o para modo IX+ :
(A) – (M);
(B) PC ← (PC) + $0002 + rel **si el resultado es $00**
o para modo SP1 :
PC ← (PC) + $0004 + rel **si el resultado es $00**

Descripción: CBEQ compara el operando con el acumulador (A) y causa una bifurcación si el resultado es cero. La instrucción CBEQ combina CMP y BEQ para rutinas de 'lookup table' más rápidas.
CBEQ IX+ compara el operando direccionado por H:X al acumulador A y causa una bifurcación si el resultado es cero. Entonces, H:X se incrementa sin tener en cuenta si se toma una bifurcación.
CBEQ IX1+ opera de la misma manera, sólo que con un desplazamiento de 8 bits, se agrega a la dirección eficaz del operando.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
CBEQ opr	DIR	31	dd rr	5
CBEQA #opr, rel	IMM	41	ii rr	4
CBEQX #opr, rel	IMM	51	ii rr	4
CBEQ X+, rel	IX+	71	rr	4
CBEQ opr, X+, rel	IX1+	61	ff rr	5
CBEQ opr, SP, rel	SP1	9E61	ff rr	6

CBEQA - Compara A con Inmediato, Bifurca si es Igual

Operación: **(A) – (M);**
PC ← (PC) + $0003 + rel **si el resultado es $00**

Descripción: CBEQ compara un operando inmediato con el acumulador (A) y causa una bifurcación si el resultado es cero. La instrucción CBEQA combina CPX y BEQ para rutinas de 'lookup table' más rápidas.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
CBEQ opr	DIR	31	dd rr	5
CBEQA #opr, rel	IMM	41	ii rr	4
CBEQX #opr, rel	IMM	51	ii rr	4
CBEQ X+, rel	IX+	71	rr	4
CBEQ opr, X+, rel	IX1+	61	ff rr	5
CBEQ opr, SP, rel	SP1	9E61	ff rr	5

CBEQX - Compara X con Inmediato, Bifurca si es Igual

Operación: **(X) – (M);**
PC ← (PC) + $0003 + rel **si el resultado es $00**

Descripción: CBEQX compara un operando inmediato con la parte baja del registro de índice (X) y causa una bifurcación si el resultado es cero. La instrucción CBEQX combina CMX y BEQ para el control de contador de lazos más rápidos.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
CBEQ opr	DIR	31	dd rr	5
CBEQA #opr, rel	IMM	41	ii rr	4
CBEQX #opr, rel	IMM	51	ii rr	4
CBEQ X+, rel	IX+	71	rr	4
CBEQ opr, X+, rel	IX1+	61	ff rr	5
CBEQ opr, SP, rel	SP1	9E61	ff rr	5

CLRH - Borra la parte alta del Registro de Índice (H)

Operación: **H ← $00**

Descripción: Los contenidos de la parte alta del registro de índice (H) se reemplaza por ceros.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
0	1	1	—	—	0	1	—

Se pone a 1 el bit Z (cero) y se pone a 0 el bit N y V.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
CLRH	INH (H)	8C		1
CLR opr, SP	SP1	9E6F	rr	4

CPHX - Compara el Registro de Índice con la Memoria

Operación: **(H:X) – (M:M + $0001)**

Descripción: CPHX compara el registro de índice (H:X) con el valor en memoria de 16 bits y pone a 1 de acuerdo con el registro de código de condición.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
↕	1	1	—	—	↕	↕	↕

V: $H7\ \overline{M15} \bullet \overline{R15} \bullet \overline{H7} \bullet M15 \bullet R15$

Se pone a 1 si desbordamiento complemento a dos, ha resultado de la operación; de lo contrario se pone a 0.

N: R15

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1}$

Se pone a 1 si el resultado es $0000; de lo contrario se pone a 0.

C: $\overline{H7}\ M15 \mid M15 \bullet R15 \mid \overline{R15} \bullet \overline{H7}$

Se pone a 1 si el valor absoluto del contenido de la memoria es mayor que el valor absoluto del registro de índice; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
CPHX #opr	IMM	65	ii ii + 1	3
CPHX opr	DIR	75	dd	4

DAA - Ajuste Decimal del Acumulador

Operación: $(A)_{10}$

Descripción: Ajusta el contenido del acumulador (A) y el estado del bit de acarreo del CCR, después de una operación BCD (decimal codificado en binario). Para que haya una suma correcta en BCD y una exacta indicación del acarreo. El estado del bit de medio acarreo del CCR afecta el funcionamiento. (Véase la tabla de Función del DAA para detalles de funcionamiento.)

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
U	1	1	—	—	↕	↕	↕

V: U
Indefinido

N: R7
Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Se pone a 1 si el resultado es $0000; de lo contrario se pone a 0.

C: Véase la tabla de Función de DAA.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
DAA	INH	72		2

Ajuste decimal del Acumulador: La tabla de Función del DAA, muestra el funcionamiento de la instrucción DAA para todas combinaciones legales de los operandos de entrada. Las columnas de la 1 a la 4 representan los resultados de los funcionamientos de la instrucción ADC o ADD, en operandos BCD. El factor de corrección en columna 5 se añade al acumulador para restaurar el resultado de un funcionamiento en dos operandos BCD a un valor válido BCD y pone a 1 o a 0 el bit C. Todos los valores están en hexadecimal.

1	2	3	4	5	6
Valor inicial del bit C	Valor de A[7:4]	Valor inicial del bit H	Valor de A[3:0]	Factor de corrección	Valor corregido del bit C
0	0-9	0	0-9	00	0
0	0-8	0	A-F	06	0
0	0-9	1	0-3	06	0
0	A-F	0	0-9	60	1
0	9-F	0	A-F	66	1
0	A-F	1	0-3	66	1
1	0-2	0	0-9	60	1
1	0-2	0	A-F	66	1
1	0-3	1	0-3	66	1

DBNZ - Decrementa y Bifurca si no es Cero

Operación: **A ← (A) - $0001**
o
M ← (M) - $0001
o
X ← (X) - $0001;
PC ← (PC) + $0003 + rel **si el resultado es 0, para DBNZ DIR o IX1**
PC ← (PC) + $0002 + rel **si el resultado es 0, para DBNZA, DBNZX, o IX**
PC ← (PC) + $0004 + rel **si el resultado es 0, para DBNZ SP1**

Descripción: Substrae uno del contenido de A, X o M; entonces bifurca usando el desplazamiento relativo, si el resultado de la substracción no es cero.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
DBNZ opr	DIR	3B	dd rr	5
DBNZA opr	INH	4B	rr	3
DBNZX opr	INH	5B	rr	3
DBNZ opr	IX	7B	rr	4
DBNZ opr	IX1	6B	rr rr	5
DBNZ opr	SP1	9E6B	rr rr	6

DIV - Divide

Operación: $A \leftarrow (H:A) \div (X)$ $H \leftarrow$ Resto

Descripción: Divide un dividendo de 16 bits sin signo contenido, en los registros encadenados, H (registro de índice alto) y (A) el acumulador, por un divisor de 8 bits contenido en el registro X (registro de índice bajo). El cociente se pone en el acumulador (A) y el divisor queda inalterado.
Un desbordamiento (cociente > $FF) o divido por cero, pone a 1 el bit C; el cociente y el resto son indeterminados.

Códigos de condición y Formula Boleana

V		H	I	N	Z	C
—	1	1	—	—	—	↕	↕

Z: $\overline{M7} \bullet \overline{M6} \bullet \overline{M5} \bullet \overline{M4} \bullet \overline{M3} \bullet \overline{M2} \bullet \overline{M1} \bullet \overline{M0}$
Se pone a 1 si el resultado (cociente) es $00; de lo contrario se pone a 0.

C: Se pone a 1 si fue intentado un divido por cero o si ocurre un desbordamiento; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
DIV	INH	52		7

LDHX - Carga el Registro de Índice con la Memoria

Operación: $H:X \leftarrow (M:M + \$0001)$

Descripción: Carga los contenidos de la posición de memoria especificada en el registro de índice (H:X). Los códigos de condición se ponen a 1 de acuerdo con el dato.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
0	1	1	—	—	↕	↕	—

V: **0**

Se pone a 0.

N: R15

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si el resultado es \$0000; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
LDHX #opr	IMM	45	ii ii	3
LDHX opr	DIR	55	dd	4

MOV - Mueve

Operación: **(M)destino ← (M)fuente**

Descripción: Mueve un byte de datos desde una dirección fuente a la dirección destino. El dato se examina cuando se mueve y los códigos de condición se ponen a 1. El dato fuente no cambia. El acumulador no es afectado.

Hay cuatro modos de direccionamiento para la instrucción MOV:

1) IMD mueve un byte inmediato a una posición de memoria directa.
2) DD mueve un byte de la posición directa a otra posición directa.
3) IX+D mueve un byte de una posición direccionada por el registro de índice (H:X) a una posición directa. H:X se incrementa después del movimiento.
4) DIX+ mueve un byte de una posición directa a una dirección por H:X. H:X se incrementa después del movimiento.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
0	1	1	—	—	↕	↕	—

V: **0**

Se pone a 0.

N: R7

Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

Se pone a 1 si el resultado es $00; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
MOV opr	IMD	6E	ii dd	4
MOV opr	DD	4E	dd dd	5
MOV opr	IX + D	7E	dd	4
MOV opr	DIX+	5E	dd	4

NSA - Cambia los “nibbles” del Acumulador

Nibbles = 4 bits

Operación: **A ← (A [3:0] : A [7:4])**

Descripción: Cambia los 4 bits (“nibbles”) más altos y más bajos del acumulador. La instrucción NSA se usa para el almacenamiento más eficaz y el uso de los operandos del decimal codificado en binario.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
NSA	INH	62		3

PSHA - Pone el Acumulador en la Pila

Operación: ↓ **(A), SP ← (SP) - $0001**

Descripción: Los contenidos del acumulador (A) se ponen en la pila (‘stack’) en la dirección contenida en el puntero de pila (SP). Entonces el puntero de pila es decrementado para apuntar a la siguiente posición disponible en la pila. Los contenidos del acumulador permanecen inalterados.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
PSHA	INH	87		2

PSHH - Pone la parte alta del Registro Índice (H) en la Pila

Operación: ↓ **(H), SP ← (SP) - $0001**

Descripción: Los contenidos de H se ponen en la pila (‘stack’) en la dirección contenida en el puntero de pila (SP). Entonces el puntero de pila es decrementado para apuntar a la siguiente posición disponible en la pila. Los contenidos de H permanecen inalterados.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
PSHH	INH	8B		2

PSHX - Pone la parte baja del Registro Índice (X) en la Pila

Operación: ↓ (X), SP ← (SP) - $0001

Descripción: Los contenidos de X se ponen en la pila (‘stack’) en la dirección contenida en el puntero de pila (SP). Entonces el puntero de pila es decrementado para apuntar a la siguiente posición disponible en la pila. Los contenidos de H permanecen inalterados.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
PSHX	INH	89		2

PULA - Saca el Acumulador de la Pila

Operación: $SP \leftarrow (SP + \$0001); \uparrow(A)$

Descripción: El puntero de pila (SP) es incrementado a la dirección del último operando en la pila (‘stack’). Entonces el acumulador (A) es cargado con los contenidos de la dirección apuntada por el SP.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
PULA	INH	86		2

PULH - Saca la parte alta del Registro Índice (H) de la Pila

Operación: **SP ← (SP + $0001); ↑ (H)**

Descripción: El puntero de pila (SP) es incrementado a la dirección del último operando en la pila. Entonces H es cargado con los contenidos de la dirección apuntada por el puntero de pila SP.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
PULH	INH	8A		2

PULX - Saca la parte baja del Registro Índice (H) de la Pila

Operación: **SP ← (SP + $0001); ↑ (X)**

Descripción: El puntero de pila (SP) es incrementado a la dirección del último operando en la pila. Entonces X es cargado con los contenidos de la dirección apuntada por el puntero de pila SP.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando(s)	
PULX	INH	88		2

STHX - Guarda el Registro de Índice

Operación: **(M:M + $0001) ← H:X**

Descripción: Guarda el registro de índice (H:X) en la posición de memoria especificada. Los códigos de condición se ponen a 1 de acuerdo con el dato.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
0	1	1	—	—	↕	↕	—

V: **0**
Se pone a 0.

N: R7
Se pone a 1 si el resultado MSB es 1; de lo contrario se pone a 0.

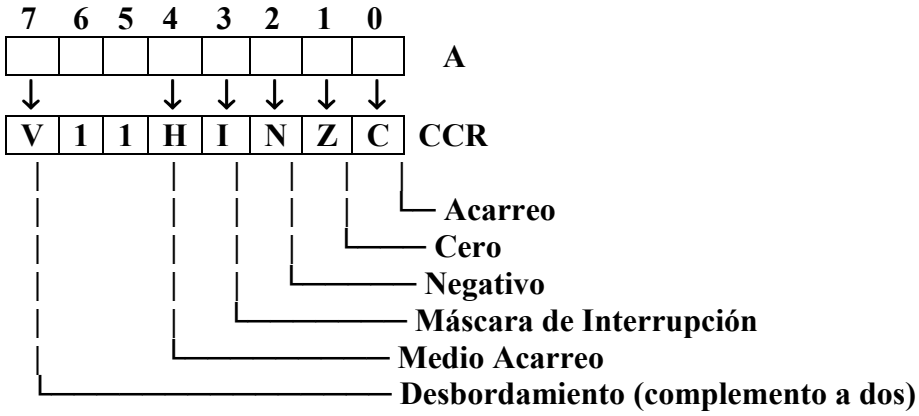
Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Se pone a 1 si el resultado es $0000; de lo contrario se pone a 0.

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
STHX opr	DIR	35	dd	4

TAP - Transfiere el Acumulador al Registro de Código de Condición

Operación: CCR ← (A)



Descripción: Transfiere los contenidos del acumulador (A) al registro de código de condición (CCR).

Códigos de condición y Formula Boleana

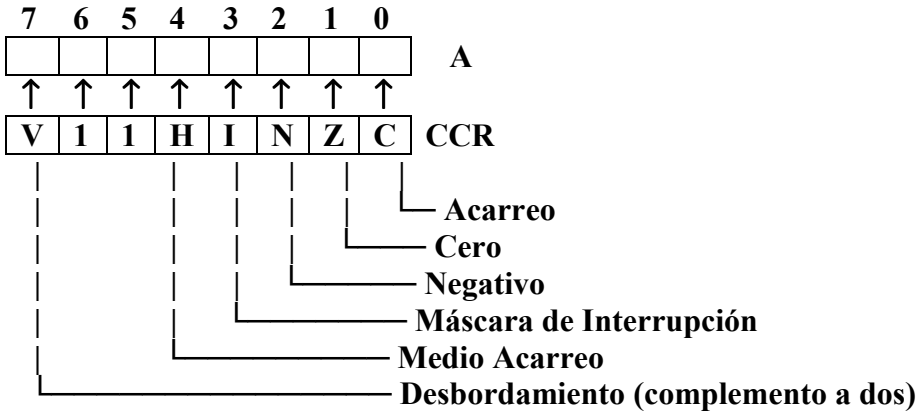
V			H	I	N	Z	C
↕	1	1	↕	↕	↕	↕	↕

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
TAP	INH	84		2

TPA - Transfiere el Registro de Código de Condición al Acumulador

Operación: (A) ← (CCR)



Descripción: Transfiere los contenidos del registro de código de condición (CCR) en el acumulador (A).

Códigos de condición y Formula Boleana

V		H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
TPA	INH	85		1

TSX - Transfiere el Puntero de Pila al Registro de Índice

Operación: **H:X ← (SP + $0001)**

Descripción: Carga el registro de índice (H:X) con 1 más el contenido del puntero de pila (SP). Los contenidos del SP no cambian. Después de una instrucción TSX, H:X apunta al último valor que fue guardado en la pila.

Códigos de condición y Formula Boleana

V		H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
TSX	INH	95		2

TXS - Transfiere el Registro de Índice al Puntero de Pila

Operación: **SP ← (H:X - $0001)**

Descripción: Carga el puntero de pila (SP) con el contenido del registro de índice (H:X) menos 1. Los contenidos del H:X no cambian.

Códigos de condición y Formula Boleana

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Ningún bit afectado

Forma, Modos de Direccionamiento, Código Máquina y Ciclos

Fuente	Modo de Direccionamiento	Código Máquina		HC08 Ciclos
		Opcode	Operando (s)	
TXS	INH	94		2

Herramientas de Desarrollo

Kits de Desarrollo ICS

Los equipos de desarrollo son un parte importante para cualquier microcontrolador y el Simulador En-Circuito o ICS, es un equipo que incluye todo lo que un diseñador necesita para iniciar un desarrollo. Proporciona simulación de software completa, simulación en-circuito con el cable para pinchar el microcontrolador usuario, emulación en-circuito de tiempo real (con un reloj fijo y un pin I/O no disponible para la emulación), un programador de la FLASH (en el zócalo de la propia placa ICS o en la placa de usuario) y un depurador en-circuito usando el BDM MON08. El BDM (Back Ground Module)

También incluye muestras, todos los cables y la fuente de alimentación y la documentación completa. El software proporcionado en este equipo puede trabajar de forma autónoma, usando el simulador y está totalmente gratuito en la web de Motorola. El kit completo **M68ICS08** está disponible a un precio de 295$.

Kits de Desarrollo Modular MMEVS y MMDS

• **KITMMEVS:** 1.495 $
Emulador tiempo real de costo efectivo

• **MMDS:** 3.950 $
Emulador tiempo real de altas prestaciones

El kit modular KITMMEVS de 1.495$ es un emulador en tiempo real completo, con emulación en-circuito para todos los tipos de microcontroladores de la familia 68HC05 y 68HC08. Basado en una placa base, un módulo de personalización, un cable y la punta de emulación. Este equipo también incluye un emulador ICS.

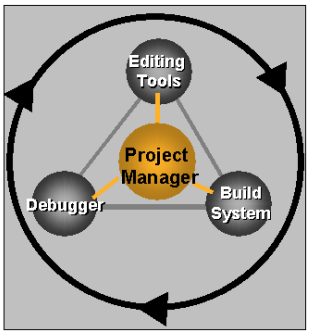
El kit modular MMDS de 3.950$ es el emulador de mayores prestaciones de Motorola, incluye todas las características y componentes del kit MMEVS y añade el analizador de estados del bus y la memoria de doble puerto. El analizador de estado del bus puede ser muy útil al poner a punto problemas muy complejos.

Todo el software está disponible en la web de Motorola, totalmente gratuita.

CodeWarrior Development Studio para la familia 68HC08

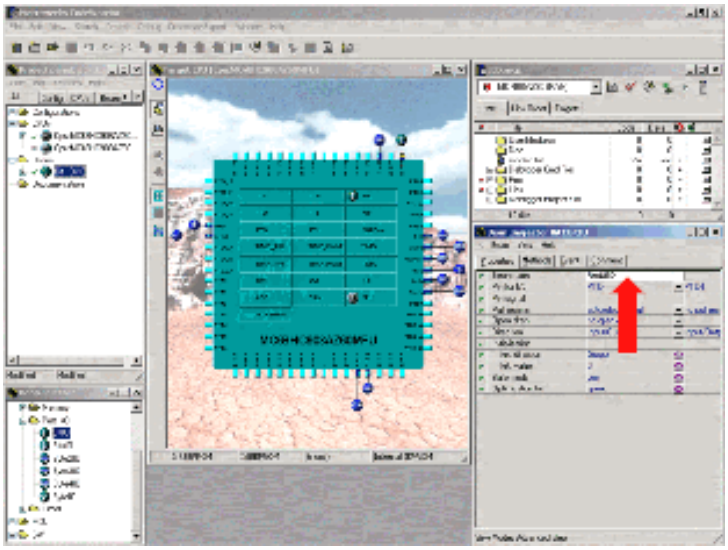
Motorola y Metrowerks rompen en el mercado del software proporcionando una nueva generación de herramientas para acelerar el tiempo de salida al mercado y mejorar calidad.

❑ CodeWarrior Development Studio for HC08, Special Edition.



¡ GRATIS EN LA WEB !

- Generador de Código con un entorno de desarrollo integrado IDE de CodeWarrior.
- Editor
- Ensamblador, Linker y Depurador a nivel fuente soportando todos los 68HC08.
- Compilador C y depurador a nivel fuente C (ANSI C limitado a 4KBytes de código)
- Simulador completo del chip y programación de la Flash con la tecnología P&E Microcomputer Systems.
- Soporte total a las herramientas de desarrollo hardware 68HC08 de Motorola 68HC08.
- Herramienta de diseño de aplicación rápida **"Processor Expert"** de Unis



❑ CodeWarrior Development Studio for HC08, Standard Edition. 1995$

Contiene lo mismo que la "Special Edition", pero sin límite de código en C.

❑ CodeWarrior Development Studio for HC08, Professional Edition. 3995$

Contiene lo mismo que la "Standard Edition", pero con numerosas funcionalidades.

CodeWarrior Standard Development Tools	Special Edition	Compiler Promo*	Standard Edition	Professional Edition
CodeWarrior IDE	Yes	Yes	Yes	Yes
Macro Assembler, Assembly Debug Set	Yes	Yes	Yes	Yes
C-Source Debug Set	4K limited	Yes	Yes	Yes
ANSI C compiler	4K limited	Yes	Yes	Yes
- Over 60 optimization strategies	Yes	Yes	Yes	Yes
- Selectable Optimization Trade-offs (Slider Bars)	Yes	Yes	Yes	Yes
Flash Programming	Yes	Yes	Yes	Yes
Simulation and Emulation Tools (P&E)	Yes	Yes	Yes	Yes
- In-Circuit Emulation (MMDS/MMEVS)	Yes	Yes	Yes	Yes
- In-Circuit Simulation/Debug/Program (ICS)	Yes	Yes	Yes	Yes
- Full-Chip Simulation (FCS)	Yes	Yes	Yes	Yes
- Cyclone (standalone programming)	Yes	Yes	Yes	Yes
- Multilink08 (in-circuit debug, programming)	Yes	Yes	Yes	Yes
Preconfigured Stationery/Code Examples	Yes	Yes	Yes	Yes
Tech Support for Hardware	Yes	Yes	Yes	Yes
Processor Expert Integration	Yes	Yes	Yes	Yes
- Standard Peripheral Beans	Yes	Yes	Yes	Yes
- Complex Peripheral Beans	-	-	-	Yes
- Access to Unis 170 SW Beans	-	-	Yes	Yes
- Bean Wizard	-	-	-	Yes
- Object Code High-Level Drivers such as LIN, CAN, USB	-	-	-	Yes
- Source Code High-Level Drivers such as LIN, CAN, USB	-	-	-	Optional Upgrades
C++/compact C++ Compiler Extension and Debug Set	-	-	Optional Upgrade ($449)	Yes
Cycle-accurate Simulator	-	-	Optional Upgrade ($595)	Yes
SoftTrace Tool (Simulator required)	-	-	Optional Upgrade ($449)	Yes
Profiling & Analysis Tools (Simulator required)	-	-	Optional Upgrade ($495)	Yes
Code Coverage Tools (Simulator required)	-	-	Optional Upgrade ($495)	Yes
Data Visualization Tools (Simulator required)	-	-	Optional Upgrade ($695)	Yes
RiMC Adapter	-	-	Optional Upgrade ($449)	Yes
Encryption Tools	-	-	Optional Upgrade ($449)	Yes
Price	No Charge	($995)	$1.995	$3.995